Technical Report 1045

Three-Dimensional Recognition of Solid Objects from a Two-Dimensional Image

Daniel P. Huttenlocher

MIT Artificial Intelligence Laboratory

Three-Dimensional Recognition of Solid Objects from a Two-Dimensional Image

Daniel Peter Huttenlocher

Submitted to the Department of Electrical Engineering and Computer Science
on April 29, 1988 in partial fulfillment of the
requirements for the degree of Doctor of Philosophy

Abstract. This thesis addresses the problem of recognizing solid objects in the three-dimensional world, using two-dimensional shape information extracted from a single image. Objects can be partly occluded and can occur in cluttered scenes. A model based approach is taken, where stored models are matched to an image. The matching problem is separated into two stages, which employ different representations of objects. The first stage uses the smallest possible number of local features to find transformations from a model to an image. This minimizes the amount of search required in recognition. The second stage uses the entire edge contour of an object to verify each transformation. This reduces the chance of finding false matches.

A new method is developed for computing transformations from a model to an image. It is shown that when perspective viewing is approximated by orthographic projection plus scale, three corresponding model and image points define a unique transformation, up to a reflection. The solution method based on this result only involves second order equations, and thus is fast and robust.

Recognizing objects under projection requires features that are relatively stable over changes in viewpoint. Stable features are obtained by segmenting edge contours at zeroes of curvature, because these points are preserved under projection. Each feature defines either a point and an orientation or three points, so only one or two features are needed to compute a transformation. Thus the number of transformations considered in recognition is only quadratic in the number of corresponding model and image features.

Thesis Co-Supervisor: W. Eric L. Grimson
Title: Matsushita Associate Professor of Computer Science and Engineering

Thesis Co-Supervisor: Shimon Ullman
Title: Professor of Psychology

Acknowledgments

I would like to thank my advisors Shimon Ullman and Eric Grimson for their guidance over the last three years. They have become good friends and valued colleagues. My third committee member, Tomás Lozano-Pérez, repeatedly challenged my thinking, and always led me to a clearer understanding of the problems I was working on.

The MIT Artificial Intelligence Laboratory is a unique research environment. I wish to thank Marvin Minsky and Patrick Winston for building and maintaining such an excellent laboratory. The many, sometimes heated, discussions with other members of the laboratory have helped me see how we might one day put all of this together.

Part of this research was done at Schlumberger in Palo Alto, and I would like especially to thank Marty Tenenbaum and Richard Lyon for their support. Discussions with people at SPAR provided me with new insights, and gave this work a broader perspective.

Without the encouragement of my friends and family, who were always there when I needed them, I doubt that this thesis could have been completed.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Army contract DACA76-85-C-0010, in part by the Office of Naval Research University Research Initiative Program under Office of Naval Research contract N00014-86-K-0685, and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research Contract N00014-85-K-0124.

Table of Contents

List of Figures

Chapter 1

Introduction

Visual recognition is the process of finding a correspondence between parts of the sensory input and stored representations of objects in the world. Many sources of information are potentially important for recognition, including shape, motion, color, texture, and shading. This thesis is primarily concerned with using two-dimensional shape information, extracted from a single black and white image, to recognize solid objects in the three-dimensional world. The objects can be partly occluded and can occur in cluttered scenes, such as the polyhedron near the center of Figure 1.

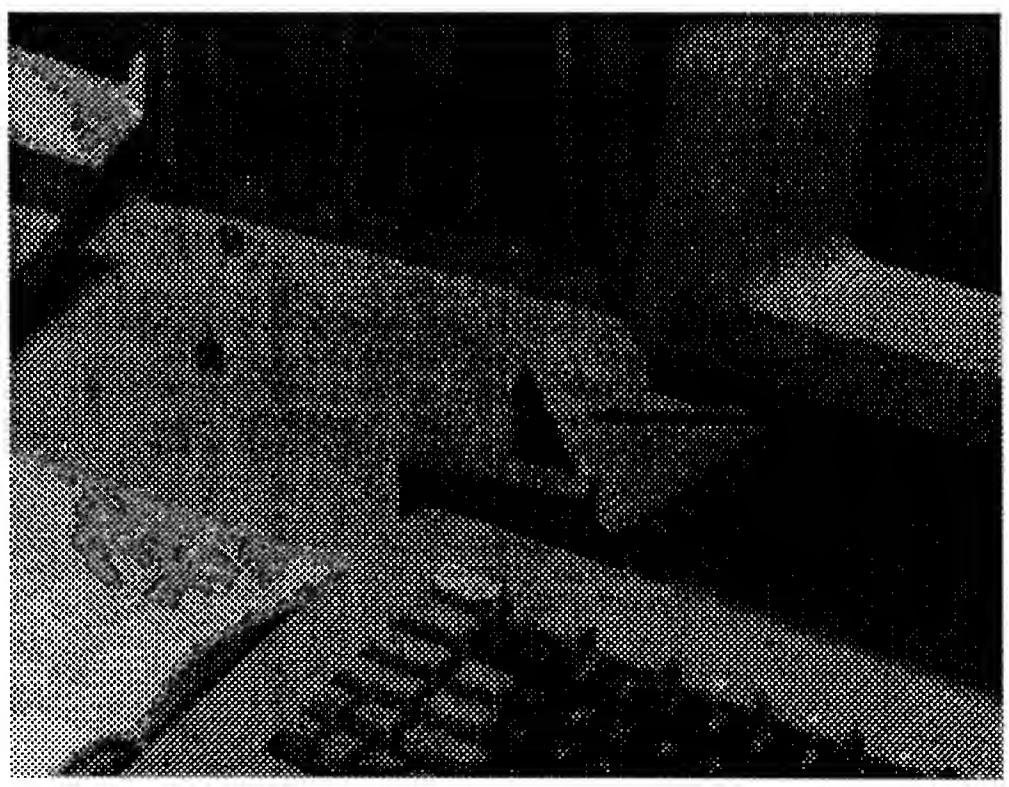

Figure 1. The type of scene of interest in this thesis.

Machine recognition systems are generally *model based*, matching geometrically accurate models of objects against an unknown image (for recent reviews see [Besl85] [Chin86]). A model specifies a set of features, or attributes of an object. Recognizing an instance of an object involves finding a consistent set of corresponding model and image features, such that some transformation maps each model feature onto its corresponding image feature. This transformation generally consists of a translation, rotation and scaling of a rigid object, and

thus specifies the position and orientation (or pose) of an object with respect to an image.

A consistent set of corresponding model and image features may not actually specify a correct match of a model to an image. Particularly if a set of corresponding features is small, it may be that the image features are accidentally positioned such that they are consistent with a valid transformation of a model. In order to reduce the likelihood of such an event, most model based recognition systems search for large sets of model and image features that are consistent with a single transformation.

The ORA (Object Recognition by Alignment, pronounced "aura") system developed in this thesis does not search for large sets of corresponding model and image features. Instead, the smallest possible sets of features are used to hypothesize potential transformations from a model to an image. Each transformation is then verified by aligning the model with the image, and comparing the aligned model edge contours with nearby image edge contours.

The central idea behind the alignment approach is to separate the matching problem into the two stages of i) finding possible transformations from a model to an image, and ii) checking those transformations by aligning the model with the image. Much of the power of the approach is derived from the fact that these two operations require relatively different representations of an object. The best representations for computing a transformation are coarse, local, and relatively sparse features. In contrast, for verification the best representations are more complete but less abstract, such as the edge contours of an object. The approach also minimizes the amount of search in recognition by using the smallest possible number of features to compute a transformation.

The alignment approach extends directly to the problem of recognizing non-rigid deformations of objects. A non-rigid transformation is approximated by a set of rigid partial matches. These rigid subparts of an object can be recovered at recognition time. In contrast, existing systems that allow for non-rigid deformations require the rigid subparts to be specified by the object models.

Some recent psychophysical data provide confirming evidence of the importance of alignment in recognition. These studies suggest that human recognition involves explicitly rotating a stored representation of an object in order to align it with an image [Tarr88]. One of the strengths of machine vision research has been the ability to combine independent computational and biological evidence for a given theory [Marr82]. For instance, the development of edge detection operators was based on both computational tractability and physiological plausibility [Marr80]. At higher levels of processing, such as recognition, there has been less success at relating biological systems to computational theories. Therefore it

is particularly encouraging to find independent support for the importance of alignment in recognition.

1.1 The Model Based Recognition Paradigm

The model based recognition problem is to find transformations that map stored models onto their instances in an image. Each transformation specifies a correspondence between a set of model features and a set of image features. The larger this set of corresponding features, the more likely it is that the transformation correctly matches a model to an instance in an image. Therefore recognition is generally cast as a search for transformations that match a large number of model features to image features.

The tasks addressed by most model based vision systems fall into one of four basic categories. It is generally assumed that objects are rigid, or composed of rigid subparts. The major differences between tasks are the dimensionality of the sensor data, of the world, and of the models. Combinations of these three parameters specify a particular rigid-body transformation from a model to an image. The four basic tasks and their transformations are given in Table 1.

Data	World	Model	Transformation
2D	2D	2D	2D similarity
3D	3D	3D	3D similarity
2D	3D	2D	approx. 2D affine
2D	3D	3D	3D-2D projection

Table 1. The four basic recognition tasks and the corresponding transformation from a model to an image.

The task in the first row of the table is two-dimensional recognition, where the mapping from a model to an image is a 2D similarity transform (a translation, rotation and scale factor). The second task is three-dimensional recognition, where the data is obtained from a 3D sensor such as a laser range finder or stereo matcher. In this case the transformation is again a similarity transform, but in three dimensions. The third task is recognition of a planar object positioned in three-space from a single two-dimensional view. Here a model and its image are in approximate 2D affine correspondence (a linear transformation plus translation).

The final task is recognizing solid objects in three-space from a single two-dimensional view (3D from 2D recognition). In this case the mapping from a

model to an image is the 2D projection of a 3D similarity transform. While 3D from 2D recognition was one of the first tasks addressed in the machine vision literature [Roberts65], only a few systems have considered the general problem [Lowe87] [Thompson87]. In addressing each of the three questions posed at the beginning of next section, it will be seen that 3D from 2D recognition is the most difficult recognition task.

1.2 Key Issues in Model Based Recognition

Within the model based recognition framework, there are three central questions:

1. What are good features for computing transformations?
2. What is the least amount of search needed to find transformations from a model to its instances an image?
3. How can it be verified that a hypothesized transformation specifies a correct match of a model to an instance?

Question one, about good features for computing a transformation, is partially answered by considering the reliability with which features can be detected. A feature should be detectable in the presence of sensor noise and partial occlusion of objects. This eliminates features that encode detailed shape information, because they are sensitive to noise, and features that depend on global properties of an object, because they are sensitive to partial occlusion. Many shape representations do not meet these criteria [Blum78] [Brady84] [Lowe85] [Bolles82]. Some representations, such as the curvature primal sketch [Asada86], address these issues for the case of two-dimensional recognition.

To complicate matters, in 3D from 2D recognition the shape of a feature in an image may be very different from the actual model shape, due to projective distortion. Thus in 3D from 2D tasks, it is important that the features used to compute a transformation be relatively stable over changes in the viewpoint of an object. Most existing shape representations are quite sensitive to changes in three-dimensional viewpoint. Chapter 5 develops a shape representation where edge contours are segmented at zeroes of curvature. These segmentation points are preserved under projection, and thus the representation is relatively stable over changes in viewpoint.

Question two, about how much search is needed to find possible transformations, depends on the number of features required to compute a transformation and on the search method. If n features are required to compute a transformation, then each n-tuple of matching model and image features may specify a different transformation. Any transformation could correspond to a correct

match, so the size of the search space is bounded below by the number of distinct transformations.

Two methods are commonly used to find transformations. The first method searches for large sets of corresponding features that are consistent with a single transformation [Bolles82] [Grimson84]. This method generally considers a given transformation more than once, and thus performs more search than may be necessary. The second method searches for large clusters of similar transformations [Thompson87] [Lamdan87]. This requires searching a large multi-dimensional clustering table. Chapter 3 formalizes the recognition problem, and analyzes the major search techniques.

The search for possible transformations is further complicated in 3D from 2D recognition tasks. First, certain search techniques [Bolles82] [Grimson84] use pruning methods that directly compare model measures with image measures. This requires the dimensionality of the sensor data and the world to be the same, which is not the case in 3D from 2D recognition. Second, it is more difficult to solve for a three-dimensional transformation from two-dimensional sensory data than from three-dimensional data. Chapter 4 presents a new method for computing a three-dimensional transformation from two-dimensional sensory data. The method requires only three corresponding model and image points, is fast to compute, and is robust with respect to noise.

Question three, on verifying a possible match, has generally been addressed by requiring that a transformation match some minimum number of model features to image features [Bolles82] [Grimson84] [Thompson87] [Lamdan87]. This is not always adequate, however, because a number of image features may be accidentally positioned such that they are consistent with a valid transformation of a model. For example, the image features on the right side of Figure 2a are a possible projection of the model features shown on the left. In Figure 2b, however, the edge contours of the model and the image show that this correspondence does not specify a correct match.

Several factors affect the likelihood of such a false match. If the individual features are highly distinctive, then an accidental match becomes unlikely. With the exception of [Bolles82], however, most recognition systems use only a few different types of features. Features that capture the entire shape of an object also make a false match unlikely, but most systems use sparse features, such as corners. 3D from 2D recognition tasks increase the likelihood of a false match, because a given set of model features can correspond to many different spatial arrangements of image features. Finally, cluttered images increase the chance of false matches. Many recognition systems have not been tested on cluttered images, and are likely to find false matches in such images. Chapter 2 reviews

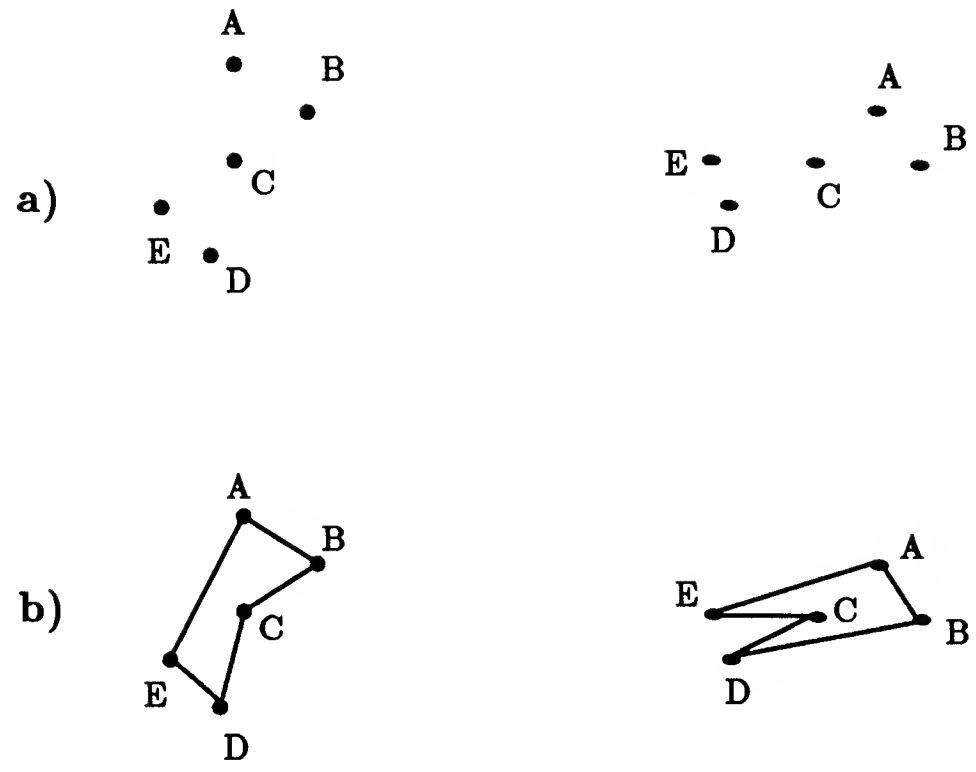

Figure 2. A set of image features that are consistent with a model, but where the edge contours reveal that the match is not correct.

several existing recognition systems in detail.

In contrast with existing model based recognition systems, the method developed in this thesis verifies a transformation by aligning a model with a hypothesized instance, and comparing the model edge contours to the image. Thus while transformations are hypothesized using sparse, simple features, verification is done by comparing an entire model with an image. Chapter 6 discusses the implementation of the verification procedure.

The three questions at the beginning of this section form the center of the investigations reported in this thesis. The remainder of this chapter presents the specific recognition task addressed in this thesis, outlines how the ORA system solves that task, and briefly summarizes the other major approaches to model based recognition.

1.3 The Recognition Task

This thesis addresses the problem of how to recognize objects that are arbitrarily positioned in three-space, from a single two-dimensional view (3D from 2D recognition). Thus an object has three translational and three rotational degrees of freedom, which must be recovered from two-dimensional sensory data. The input to the recognizer is a grey-level image and a set of three-dimensional models. The output is all of the transformations that map a model onto an

instance in the image.

The imaging process is assumed to be well approximated by the "weak perspective" model presented in the next section. This imaging model approximates perspective projection by orthographic projection plus a scale factor. It is further assumed that the transformation from an object to an image is rigid, or is composed of a set of locally rigid parts. Unlike existing recognition systems, however, the rigid parts of an object need not be specified by an object model. In Chapter 7 two methods are presented for recovering the rigid subparts of a transformation at recognition time. This allows recognition of objects that have been bent or stretched in some unexpected manner, as well as objects that can articulate at certain predefined points.

Figure 3. An object that is not well described by the shape of its occluding contours.

Recognition is restricted to objects that can be identified by the shape of their edge contours. Thus an object must have a canonical shape, and must have occluding contours that are relatively stable over small changes in viewpoint. This is true of most objects that do not have smoothly changing surfaces. Objects such as the sculpture in Figure 3 do not meet these criteria, however, because the shape of the occluding contours changes substantially for slight differences in viewpoint. Some current work addresses the problem of how to apply the alignment method of recognition to objects that have smoothly changing surfaces [Basri88].

There are relatively few constraints on the kinds of scenes in which objects can be recognized. An object can be partly occluded and highly foreshortened, and the scene can be cluttered. The major limitation on clutter is the performance of the edge detector on images that have many intensity edges in close proximity. The level of image complexity of interest is illustrated in Figure 1, where the object to be recognized is the polyhedron near the center of the scene.

1.4 The Imaging Model

In order to recover three-dimensional information from a two-dimensional image, a recognition system must model the projection that occurs in the imaging process. The type of imaging model influences the computational complexity of the matching algorithm, because it determines how much information is needed to solve for a transformation from a model to an image. The most accurate model of the imaging process is perspective projection. As illustrated in Figure 4, under this model the viewing axis, **v** is perpendicular to the image plane, I, and intersects that plane at a point, o. The center of projection, f, is a point along **v** such that each point in the world, p, is connected to its image, p', in I, by a ray passing through f. Two parameters of the perspective model are the focal length, which is the distance $|f - o|$, and the location of o in image coordinates.

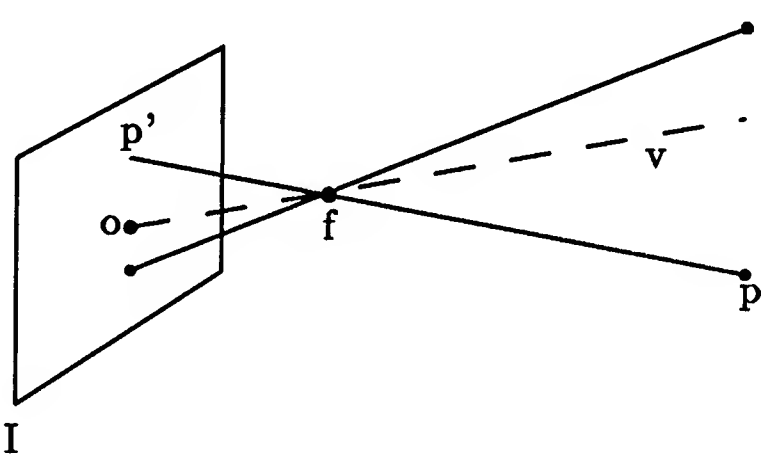

Figure 4. Perspective projection.

Throughout this thesis a coordinate system with the origin in the image plane, I, and the z-axis perpendicular to I is assumed. In the perspective model, the origin is at o, and the z-axis is along **v**.

While perspective projection is an accurate model of imaging, it is relatively difficult to recover the position and orientation of an object from its image under perspective projection. In general, computing a transformation requires six model points and six corresponding image points [Fischler81]. The equations for solving this problem are relatively unstable, and the most successful methods use more than six points and an error minimization procedure such as least squares [Lowe87].

If the camera parameters (the focal length and the location of o in image coordinates) are known, then three corresponding model and image points specify up to four possible transformations from a model plane to the image plane [Fischler81]. A solid object may be reflected about a model plane, resulting in a total of up to eight possible transformations. There are two drawbacks to this

method of computing transformations. First, a calibration operation is required to determine the camera parameters. If the camera has a variable focal length it is necessary to recalibrate whenever the focal length is changed. Second, the solution method is complicated and involves solving a quartic equation, so it is relatively sensitive to noise.

In contrast to perspective projection, under orthographic (or parallel) projection, each point, p, in the world is connected to its image, p', in the image plane, I, by a ray, P, which is perpendicular to I, as shown in Figure 5. The major difference between perspective and orthographic projection is that under perspective projection objects appear smaller when they are farther away. Thus, if a linear scale factor is added to orthographic projection, a relatively good approximation to perspective is obtained. The approximation becomes poor when an object is deep with respect to the field of view, because a single scale factor cannot be used for the entire object [Horn86] [Thompson87]. For instance, railroad tracks going off towards the horizon, or objects viewed from very close up are not well approximated by this model. Orthographic projection plus scale has been termed "weak perspective", because it approximates perspective well under most viewing conditions.

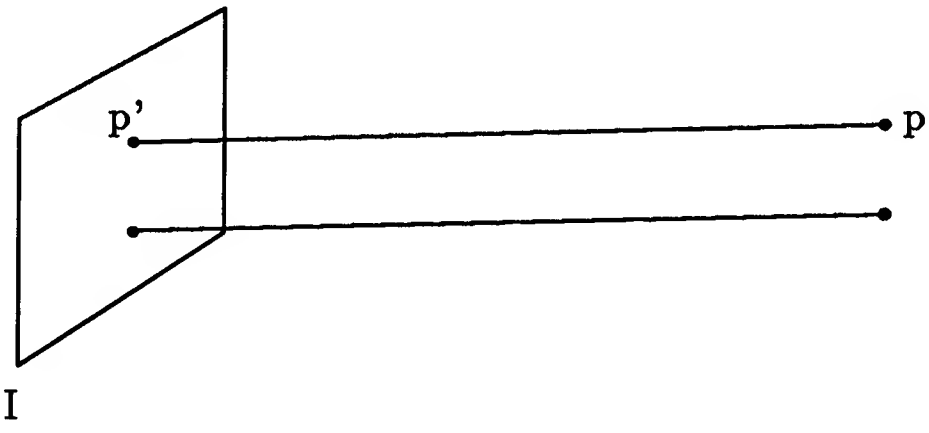

Figure 5. Orthographic projection.

Under orthographic projection all z-positions are equivalent, so under the weak perspective model there are only two translations, in x and y in the image plane. Thus, the six degrees of freedom of an object are: a two-dimensional translation, a three-dimensional rotation, and a scale factor that depends on the distance to and size of the object. Such a transformation, which preserves relative lengths, is called a similarity transform. In three dimensions, a similarity transform, T can be represented as a 3×3 orthonormal (rotation) matrix, $\mathbf{R}$, a three-dimensional translation vector, $\mathbf{t}$, and a scalar, s, such that $T(\mathbf{x}) = s\mathbf{R}\mathbf{x} + \mathbf{t}$, for any three-dimensional vector, $\mathbf{x}$.

The image coordinates of a model point $\mathbf{x}$ under weak perspective are (x, y), where $(x, y, z) = T(\mathbf{x})$, and the similarity transformation, T, has a translation

vector, **t**, that is zero in its z-coordinate. In Chapter 4 it is shown that this transformation can be computed, up to a reflective ambiguity, from three corresponding model and image points. The method only involves solving a second order system of two equations, and is relatively robust with respect to noise. Due to the computational simplicity of this transformation, the ORA system uses the weak perspective imaging model.

A number of recognition systems [Brooks81a] [Cyganski85] [Lamdan87] [Thompson87] have used the weak perspective imaging model. Unlike the current work, however, these systems have not solved the problem of how to compute a three-dimensional transformation directly from corresponding model and image points. Instead, they either use a heuristic approach to finding a transformation [Brooks81a], store views of an object from all possible angles in order to approximate the transformation by table lookup [Thompson87], or restrict recognition to planar objects [Augusteijn86] [Cyganski85] [Lamdan87].

1.5 The Alignment Approach

In this thesis the recognition problem is decomposed into three stages: (i) feature extraction and perceptual grouping, (ii) solving for transformations that align models with an image, and (iii) comparing aligned models with an image. This section briefly describes each of the three stages of processing in the ORA system.

Initially an image is processed to extract edges [Canny86], and the edge pixels are chained into contours. The local curvature of the contours is computed, and zeroes of curvature are used as segmentation points. The resulting edge segments are relatively stable over changes in position and orientation, because zeroes of curvature are preserved under projection. Locally connected groups of these edge segments are used to define features for alignment. Each feature contains either a point and an orientation, two points and two orientations, or three points.

Pairs or singletons of corresponding model and image points are used to compute possible transformations from a model to an image, using the method developed in Chapter 4. Only one or two corresponding model and image features are needed to specify a transformation that is unique up to a reflection. Each transformation is then used to align the model with the image, by transforming the model into image coordinates.

A transformation is verified by comparing the aligned model edges with image edges. Positive evidence, such as proximity and cotermination of model and image edges, is used to find edge contours that are accounted for by a match. Negative evidence, such as crossing model and image edges, is used to eliminate

model edge contours that are not accounted for. If more than a certain percentage of a model's edge contours are matched, then a transformation is accepted as specifying a correct match of the model to the image. The verification process is hierarchical. First local model and image points are compared, and then the entire edge contours are traced. This speeds up the verification process, because many matches can be disqualified with a simple check.

The ORA system finds all transformations that map a substantial portion of a model's edges onto part of an image. In contrast, many existing recognition systems find only the best match, or the few best matches [Lamdan87] [Thompson87] [Linainmaa85]. Finding the best matches implicitly assumes that there are one or more instances of an object in the image. Such systems do not really perform recognition, because it is never decided whether or not an instance is present.

The central idea underlying the alignment approach is to separate the matching problem into the two stages of i) finding possible transformations from a model to an image, and ii) verifying those transformations by aligning the model with the image. The reason for having distinct stages is that the two operations require relatively different representations of an object. The best representations for computing a transformation are coarse, local, sparse features. Features that are local and coarse are more reliably detectable than are fine-scale or global features. Sparse features reduce the number of possible matches between a model and an image. The best representations for verification, on the other hand, are more complete but less abstract descriptions, such as the edge contours of an object. A complete description of an object may be needed in order to decide whether or not a transformation is correct.

The secondary ideas behind approach are to reduce the amount of search in recognition, and to simplify the problem of representing objects. By identifying the smallest sets of features that are needed to hypothesize a transformation, the method minimizes the number of transformations that must be considered. By representing objects as edge contours and features extracted from edges, the models are very similar to the sensory data. This simplifies the problems of forming models and matching models to images.

The basic alignment method described in this section has been extended to the case of non-rigid deformations. A number of recognition systems allow objects to be represented as a set of rigid parts [Grimson87b] [Brooks81a]. The alignment method similarly approximates a non-rigid deformation by a set of locally rigid transformations. The difference is that the rigid parts of an object can be recovered at recognition time, rather than requiring them to be specified by the model. In Chapter 7, two different methods are presented for recovering

the rigid subparts during the matching process.

1.6 Other Approaches

Early recognition systems addressed the 3D from 2D recognition problem. The pioneering work of Roberts [Roberts65] involved the development of a system for recognizing polyhedral objects in a line drawing or photograph. The major drawback of this system, and of the later ACRONYM recognition system [Brooks81a], was that a lack of formal specification of the matching process made it difficult to determine whether failures were due to the feature extraction or to the matching process. More recent work has formalized the recognition problem as a search process [Bolles82] [Grimson84]. Within this search framework relatively little work has addressed the 3D from 2D recognition problem (e.g., [Lowe87] [Thompson87]), largely because many of the measures used to compare models with images are not preserved under projection.

The recognition problem is generally broken down into three stages: (i) extracting features from an image, (ii) selecting possible models from an object library, and (iii) matching the models to the image. The model selection problem is generally not addressed, and only one model is matched to the image at a time (some notable exceptions are [Kalvin86] [Ettinger88]). Most of the research effort in machine recognition has focused on the matching problem, determining potential transformations that map a model onto an instance in an image.

A number of methods have been proposed for finding possible transformations. The approaches can be divided into four major categories: (i) non-correspondence matching, (ii) pruned search of the exponential space of corresponding model and image features, (iii) parallel relaxation matching, and (iv) searching the space of possible transformations, either by clustering or by explicitly considering the transformations. Non-correspondence matching compares global model and image descriptors, such as moments of inertia [Teague80]. Pruned search uses relations between pairs of local model and image features to limit the exponential space of possible corresponding features [Bolles82] [Grimson84] [Lowe85]. Parallel relaxation matching is an approximation to the pruned search for corresponding sets of features [Rosenfeld76] [Davis79]. The space of transformations is either searched using the generalized Hough transform to find clusters of similar transformations [Silberberg86] [Thompson87], or by explicitly checking transformations [Augusteijn86] [Fischler81]. Chapter 2 reviews certain key recognition systems in each these classes.

The method developed in this thesis uses small sets of corresponding features to hypothesize possible transformations, and then verifies each transfor-

mation. Thus like method (iv) the search is over the set of possible transformations. Unlike existing recognition methods, however, the verification procedure compares an entire object with an image rather than requiring large groups of matching model and image features. The current approach also identifies the smallest possible number of features needed to compute each transformation.

1.7 Roadmap

The next chapter discusses some of the research that is relevant to the problems addressed in this thesis. Several recognition systems are considered, divided according to the four major methods used to match a model to an image: non-correspondence matching, pruned exponential search, parallel relaxation, and search of the transformation space. In addition to discussing recognition systems, some shape representations are considered. Most of these representations are concerned with describing objects, rather than recognizing them, and thus have somewhat different goals than the representation developed here.

Chapter 3 formalizes recognition as a search process, and compares the combinatorics of the two major methods used to structure the search. The first method considers the exponential space of corresponding model and image features, and uses pairwise relations between features to prune the search. The second method considers the possible transformations from a model to an image. This search space is polynomial, where the degree of the polynomial depends on the method used to compute a transformation. Clusters of similar transformations are found using the generalized Hough transform. By modeling this clustering technique as an occupancy problem, its limitations for 3D from 2D recognition tasks and cluttered images are shown.

Chapter 4 shows that under the weak perspective imaging model, three corresponding model and image points uniquely define a transformation (up to a reflection) that maps a solid object in three-space onto an image. This result is established in two steps. It is relatively well known that three points uniquely define a two-dimensional affine transform. It is shown here that a two-dimensional affine transform uniquely defines a three-dimensional similarity transform, up to a reflective ambiguity. A partial version of this result has also been shown in the shape from texture literature [Kanade83]. Based on the result, a method is developed to compute a transformation using three corresponding model and image points, or two corresponding points and orientations. The method involves only second order equations, and is thus fast and robust. Some estimates of the transformation error are also derived.

Chapter 5 discusses the problems of feature extraction and grouping for

recognition by alignment. A shape representation that is stable over viewpoints is obtained by segmenting edge contours at inflection points and at the ends of straight regions. These points are zeroes of curvature, which are preserved under projection. The edge segments are then used to form primitive features such as straight edges, arcs and corners. Methods are presented for robustly computing local orientation, local curvature and zero crossings from noisy data.

Chapter 6 describes the ORA (Object Recognition by Alignment) system. Edge contours are segmented at zeroes of curvature in order to yield a representation that is stable over changes in viewpoint. Groups of connected local edge segments define features for alignment. Singletons or pairs of corresponding model and image features are used to hypothesize transformations from a model to an image. These hypotheses are then verified by comparing the transformed model edges with image edges. The verification procedure is hierarchical, and requires a good correspondence between model and image edge contours. Each possible alignment of a model with an image can be computed independently of the others, and thus the technique is well suited for implementation on a massively parallel computer such as the connection machine.

Chapter 7 extends the basic alignment method to the recognition of non-rigidly deformed objects. A non-rigid deformation is approximated by a set of locally rigid alignments. The rigid parts of an object can either be specified by the object model, or can be recovered at recognition time. Two methods for determining the rigid parts of an object are presented. The first method is iterative, starting with a coarse match and refining it. The second method combines partial matches that are consistent with a single instance of an object.

Chapter 8 discusses some psychophysical results indicating that human recognition involves alignment of a stored representation of an object with an image. This data supports the separation of matching into distinct stages of alignment and comparison. The results also suggest that people form multiple representations of an object, and align an unknown instance with the stored representation that is at the most similar orientation.

1.8 Major Contributions

This section briefly summarizes the main differences between the method developed here and other recognition methods, and lists the most important results reported in the thesis.

- The recognition method developed in this thesis separates matching into the two stages of: i) solving for an alignment transformation, and ii) comparing an aligned model with an image.

- Separating matching into alignment and verification stages allows different information to be used for each stage. Coarse-scale, local, sparse features are the best representation for hypothesizing a transformation. In contrast, the best representation for verification is a more complete but less abstract description, such as the edge contours.
- Rather than searching for large sets of corresponding model and image features, the method developed here uses the smallest possible sets of features to solve for possible transformations, thus minimizing the size of the search space.
- It is shown that three corresponding model and image points determine a unique (up to a reflection) transformation mapping a solid model in three-space onto a two-dimensional image, assuming the weak perspective imaging model.
- A simple, fast, robust method is developed for computing a transformation using three matching model and image points, or two points and two orientations.
- A shape representation is developed that is relatively insensitive to partial occlusion and stable over different viewpoints. Sensitivity to occlusion is minimized by using local features. Stability over viewpoints is obtained by segmenting edge contours at zeroes of curvature, which are preserved under projection.
- The process of verifying whether or not a transformation is correct compares aligned model edges with image edges, and uses both positive and negative evidence of a match.
- Non-rigid deformations of an object are approximated by a set of local rigid alignments. The rigid subparts can either be specified as part of the model, or can be recovered dynamically at recognition time.
- Modeling the generalized Hough transform as an occupancy problem shows the limitations of this clustering technique for complex recognition tasks.
- Recent psychophysical studies provide independent confirmation of the importance of alignment in recognition.

Chapter 2

Review Of Existing Work

Computational vision is generally divided into low-level, middle-level and high-level tasks. Low-level vision is concerned with extracting primitive events, such as image discontinuities, from the sensory data, usually a grey-level image. Middle-level vision involves forming a representation of a scene, usually in terms of shape or texture primitives. High-level vision interprets a description of a scene for tasks such as recognition and obstacle detection.

This chapter discusses existing work on middle- and high-level vision that is relevant to the 3D from 2D recognition task addressed in this thesis. Certain key recognition systems and shape representations are considered. A brief description of each method or system is presented, together with an analysis of the extent to which the results are useful for 3D from 2D recognition. For a more general review of recognition work see [Besl85] [Chin86].

2.1 Object Recognition Systems

Machine recognition systems structure the recognition problem as one of matching models to an image. The matching process involves determining how models should be transformed in order to bring them into correspondence with an image. Recognition systems can be divided into two classes according to how possible transformations are determined. Systems in the first class compute global properties of an image (such as moments of inertia) and use matching model and image properties to solve for possible transformations. These systems generally cannot recognize scenes with multiple objects or partially occluded objects, because of the dependence on global image properties.

Recognition systems in the second class use corresponding model and image features to recover possible transformations from a model to an image. These systems can generally handle multiple objects in an image, as well as partially visible objects, because the features are local. A single feature generally does not contain enough information to solve for a transformation, however, so groups of features must be used. Thus systems in this class may need to perform a substantial amount of search in order to recover a transformation.

Three major methods have been developed for using corresponding model and image features to find transformations from a model to an image. These

methods are: (i) pruned search, where the exponential set of possible corresponding model and image features are considered, but pairwise constraints are used to limit the search, (ii) parallel relaxation, where a suboptimal local procedure is used to find possible corresponding sets of features in polynomial time, and (iii) searching the polynomial space of possible transformations from a model to an image.

Many recognition systems address tasks where the dimensionality of the sensory data and the world are the same. These systems generally rely on the ability to directly compare model and sensor quantities such as distances and angles, that are not preserved under projection. Thus such systems cannot be applied to 3D from 2D tasks in any straightforward manner. The discussion of each system points out any such limitations for 3D from 2D tasks.

The remainder of this section is divided according to the four major classes of systems just identified: non-correspondence matching, pruned exponential search, parallel relaxation, and search over transformations. The problem of selecting models from a large object library is also briefly considered.

2.1.1 Non-Correspondence Matching

Non-correspondence matching involves finding a transformation from a model to an image without first determining the correspondence between individual parts or features of the model and the image. Instead, global features of a model and an image are used to compute a transformation. The major limitation of the approach is that an object must be completely visible in an image, and the object must be isolated from other objects. That is, either there may only be one object in the image, or the image must be segmented into "subimages" of isolated objects. Thus non-correspondence methods are not well suited to the recognition of complex scenes with multiple overlapping objects.

Most non-correspondence approaches to recognition involve computing a parameter vector from an image, and then comparing the image vector to model vectors in order to find the best matching model. The parameters are generally functions such as Fourier transforms or moments of inertia computed from the entire image. Objects must generally lie in a known plane, so that the recognition task is two-dimensional. Much of the work on non-correspondence methods has focused on the problem of identifying parameters that are invariant to planar translation, rotation, and scale of an object [Teague80]. Non-correspondence matching using parameter vectors has also been applied to recognizing objects with three-dimensional positional freedom using three-dimensional data [Sadaji80].

A different approach to non-correspondence matching has been investigated by Cyganski and Orr [Cyganski85]. They consider the task of recognizing planar objects with arbitrary three-dimensional position and orientation from a two-dimensional view. A two-dimensional affine transformation mapping a model plane to an image plane is solved for using tensors computed from the image. An affine transform $T : \mathbf{x} \rightarrow \mathbf{x}'$ can be represented by a nonsingular $n \times n$ matrix $\mathbf{L}$ and a vector $\mathbf{b}$ such that $\mathbf{x}' = \mathbf{L}\mathbf{x} + \mathbf{b}$ for any n-dimensional vector $\mathbf{x}$. The transformation can represent translation, rotation, scaling and shearing from one plane to another. The tensors used for computing the transformation are based on moments of inertia, which are global measures of the image. Thus, like other non-correspondence approaches, the method is only applicable to recognizing totally visible objects, in images containing an isolated object.

Cyganski and Orr also present a method for recovering the three-dimensional position and orientation of a model plane with respect to an image, given the affine transform mapping the model to the image. First the model is translated in x and y according to the translational component, $\mathbf{b}$, of the affine transform. A scale factor is used to model the z translation, assuming that the absolute size of the object is known. The scale factor, K, and the rotations α, β, and γ about the x, y, and z axes, respectively, are computed from the linear transformation matrix, $\mathbf{L}$. The derivation is not presented, and no argument or proof is given that a solution necessarily exists, or is unique.

The method, however, does not yield the correct scale or orientation. For example, with $\mathbf{L}$ equal to the 2×2 identity matrix, the three rotations α, β, and γ should all be 0, and the scale factor, K, should be 1. The equations do give $\alpha = 0$, but the scale factor K is also 0. The second two rotations are then undefined, because of division by zero. Even in cases where $K \neq 0$, the computation of K is still incorrect, and hence the second two rotations which depend on K are also incorrect. The derivation of the equations is not presented or explained, so it is difficult to determine where the problem lies. The experimental results shown in the paper do not recover three-dimensional position and orientation, so apparently the method was not implemented.

2.1.2 Pruned Search

A number of object recognition systems use pruned search to hypothesize potential transformations from a model to an image. These systems start by finding possible matching pairs of model and image features, and then search for sets of these pairs that are consistent with a single position and orientation of a rigid object. In the last few years, there has been a major shift in how systems of this class search for possible correspondences. Earlier systems used a set of rules

or heuristics to try certain (generally local) combinations of features in order to find correspondences. These search techniques were incomplete, in that only certain correspondences were considered. More recent work has formalized the search process, and has developed pruning techniques that can guarantee that a correct solution will not be missed (within a specified error range). These methods do not search the entire space, which is exponential in the number of model and image features, because they prune away locally impossible combinations of features.

Heuristic Pruning

The pioneering work of Roberts [Roberts65] involved the development of a system for recognizing polyhedral objects in a line drawing or photograph. The major limitation of the system was its reliance on finding complete convex polygons, which is unrealistic both in the face of noise and occlusion. The recognizer recovered the three-dimensional position and orientation of a polyhedron using quadruples of corresponding model and image features, under an imaging model similar to weak perspective. The features were vertices, which were classified according to the number of sides in the visible polygons to which the vertex belonged. For instance, each vertex of a cube would be labeled $(4, 4, 4)$, because it belongs to three faces of four edges each.

Roberts proposed a relatively complex imaging model, where an object had 15 possible degrees of freedom: 3 translations, 3 rotations, 3 scale factors, 3 skew deformations and 3 perspective deformations. In actuality, he used a simpler model where perspective deformation was not allowed. Furthermore, skew deformations were found to be numerically unstable and were thus also eliminated. This left 2 translations (because z position is undetectable under orthographic projection), 3 rotations and 3 scale factors. Using an equation counting argument, he argued that four or more point correspondences are sufficient to solve for this because there are eight degrees of freedom. There are eight nonlinear equations in eight unknowns, however, so a more sophisticated argument is necessary to guarantee the existence and uniqueness of a solution.

The system searched for four pairs of corresponding model and image points by starting with a given pair of model and image vertices, and then following edges in the model and the image to form additional vertex pairs. In other words, the heuristic for limiting the space of possible correspondences was to consider only those sets of vertices connected by edges. This is why the system depended critically on finding closed polygons in the image. Once a consistent set of four points was found, the model was transformed and it was verified that the model points all fell within the object's external boundary. If there were more than four corresponding point pairs, a least squares method was used to

determine a more accurate estimate of position and orientation.

After Roberts, recognition work moved away from considering real images, and instead focused on line labeling, abstract shape representations, and recognizing hand segmented scenes [Marr78] [Barrow76]. Systems that worked with real images tended to address lower level tasks such as edge detection [Marr80].

ACRONYM [Brooks81a] was the first recognition system that was designed to operate on noisy and incomplete image representations. Previous systems had assumed that a good line drawing or an accurate segmentation of an image could be obtained by lower level processing. Repeated attempts to develop reliable algorithms for these tasks led Brooks to conclude that dealing with noisy image data was an inherent part of the recognition task.

ACRONYM used the weak perspective imaging model of orthographic projection plus a scale factor. The system had parameterized models that could represent classes of objects differing in the size, position, or orientation of subparts. The models were volumetric, being composed of generalized cylinders [Nevatia77]. A generalized cylinder is a specification of volume given by a spine function and a cross-sectional sweeping function. The underlying philosophy of the system was to predict how a generalized cylinder model would appear in an image given a partial restriction on position and orientation. The prediction was then compared with the image and used to further constrain the position and orientation estimate.

Like Roberts' system, ACRONYM performed an incomplete search, using certain pairs of model and image features to predict the position and orientation of other features in the image. If the predicted features were found, they were used to refine the estimate of the position and orientation of the object. An initial hypothesized match of a model feature with an image feature was used to constrain parameters of the model, such as its position, orientation, size, etc. Then additional features were used to tighten the constraints, or to discard the hypothesis if the constraints were inconsistent.

The constraints used in ACRONYM were conjunctions of algebraic inequalities. These sets of constraints were not solved, but rather were used to limit the possible configurations of the model. Thus, the constraint manipulation component was designed to decide whether a set of constraints was satisfiable, and to estimate bounds on the values of the expressions. The decision procedure for determining the satisfiability of a set of constraints was only partial, and it was not shown which classes of inconsistent constraints were undetectable. The decision procedure was also exponential in the number of variables in the constraint equations.

The procedure for estimating bounds on the values of expressions, such as

position and orientation, often produced very loose estimates because of the difficulty of solving certain equations. For instance, projection of objects rotated out of the plane involves coupled sines and cosines that were treated independently and thus gave very pessimistic bounds. Composition of unknown rotations could not be estimated at all. Thus the system was limited in its ability to recognize objects with a full six degrees of positional freedom. In fact, the system was evaluated using aerial photographs, which are essentially two-dimensional in nature.

Goad [Goad86] describes a recognition system similar to ACRONYM, in that certain model features are matched to image features and used to constrain the possible positions of other model features in the image. The system differs from ACRONYM in two significant ways. First, quantitative limits on possible positions and orientations were computed, as opposed to ACRONYM's symbolic qualitative solution. Second, the possible positions and orientations of each model feature were pre-computed for each range of positions and orientations of every other model feature. These values were stored in a table, and used at recognition time to quickly check if a given model and image feature pair was consistent with other pairs.

In addition to using precomputed values, a set of search trees were precompiled for recognizing an object given an initial matching pair of features. These trees specified which model and image feature pairs to look for next, as well as their possible positions and orientations. Thus substantial offline effort was used to save time during recognition.

Goad made several simplifying assumptions about the recognition task. The object to be recognized had to be totally visible, and there could be very little foreshortening. The latter limitation was due to the assumption that lengths do not change with rotation out of the image plane. Thus an object was limited to undergo basically two-dimensional rotation and translation. This was necessary in order to allow distances and angles measured in the 2D image to be compared with distances and angles in the 3D model, before having recovered the three-dimensional orientation of the object. Finally, the distance to the camera had to be known, at least approximately (within a factor of 2).

Local Constraints

The Local Feature Focus (LFF) system [Bolles82] was the first recognizer to express recognition as a complete search, with no heuristic rules for limiting the space of possibilities. Given a set of pairs of model and image features, $S = \{(m_j, i_k)\}$, the task is to find the largest subset of S that is consistent with a single position and orientation of the model. In other words, the largest set of features, $C \subseteq S$, for which there is a single transformation, T, that maps each

model feature onto its corresponding image feature, $C = \{(m_j, i_k)|T(m_j) = i_k\}$.

Thus, unlike previous systems, LFF's search technique was guaranteed not to miss a correct solution. LFF used a small number of distinctive features to perform two-dimensional recognition of overlapping and partially visible objects. A later system, 3DPO [Bolles86], used the same method and a larger feature set to do 3D from 3D recognition. The two systems use basically the same method, so they will be considered as one.

LFF formed a graph with each pair of model and image features, (m_j, i_k), corresponding to a node in the graph. Two nodes in the graph were connected by an arc if they specified feature pairs that were consistent with a single position and orientation of the model. Thus all $|S| \times |S|$ feature pairs were considered, and those pairs that were consistent formed arcs of the graph. The maximum clique of this graph was then computed to find the largest consistent set of model and image features. This set of features was further checked to ensure that it was globally consistent with a single transformation, T, because the graph only encodes pairwise consistency among the nodes. If a sufficiently large feature set was found to be globally consistent, then the model was successfully matched to the image.

There are two major ideas underlying LFF: to use local features with relatively distinctive labels, and to find the largest sets of model and image features that are pairwise consistent with a single position and orientation of a rigid object. Local features were used to minimize problems with occlusion and partial visibility. Each feature was labeled using a cluster of nearby features, in order to limit the number of matching model and image features.

Certain distinguished features were chosen as focus features, and then clusters of neighboring features were used to label these focus features. The labels were of the form "a corner with a neighboring hole in direction **v** at distance d". In an image with nearby or occluding objects, however, this grouping technique can produce feature clusters that come from multiple objects. Such clusters will not, in general, correctly match model clusters. Thus the use of proximity for grouping can cause a correct match to be missed.

After extracting image features, the set of matching features, S, was formed by pairing together model and image features having the same label. The graph structure was then formed with one node for each pair in S, and with pairwise consistent nodes connected by arcs.

The consistency of two nodes was checked by comparing the distance and angle between the two model features with the values for the two image features. If the differences were within an allowable error range, then the two nodes might correspond to a single position and orientation of an object. Consider two model

features each defining a point, a_m and b_m, and an orientation vector, $\mathbf{a}_m$ and $\mathbf{b}_m$, respectively, and two corresponding image features defining the points a_i and b_i with orientation vectors $\mathbf{a}_i$ and $\mathbf{b}_i$. In order for the two pairs of corresponding features to be consistent, the distance $|a_i - b_i|$ must be nearly equal to $|a_m - b_m|$. Similarly the angle between $\mathbf{a}_i$ and $\mathbf{b}_i$ must be nearly the same as the angle between $\mathbf{a}_m$ and $\mathbf{b}_m$. The allowable distance and angle ranges between each pair of model features were precomputed, so that consistency could be checked by table lookup.

One limitation of the technique is that distances and angles measured in the sensory data must be comparable with those in the model. Thus the measures used are not applicable to 3D from 2D recognition, where there is projection from the world into the image.

A maximum clique of the graph corresponds to the largest set of model and image features that are pairwise consistent with a single position and orientation of a rigid object. The maximum clique problem is NP-complete, so all known solutions are exponential in the size of the graph. Experiments with LFF and 3DPO, however, demonstrate that the graphs are sparse enough that the search time is not prohibitive for moderately complex images.

Grimson and Lozano-Pérez [Grimson84] [Grimson87a] have developed a recognition system called RAF (Recognition and Attitude Finder), that is similar to LFF, in that it searches for a maximally consistent set of model and image feature pairs. Like LFF, the RAF system uses pairwise distance and angle relations between features. The relations are the distance between features, the angle between features, and the components of the distance in the directions of orthonormal basis vectors. Grimson and Lozano-Pérez have shown that these relations contain all the pairwise distance and angle information relating two features. In other words, any other pairwise distance and angle relations can be expressed in terms of this set.

Unlike LFF, the RAF system does not use information about the identity of features to form pairs of model and image features. Instead, each model feature is paired with each image feature. The philosophy of the RAF system is that no reliable information about feature identity can be extracted from an image, only information about the geometric relations between features is useful.

RAF structures the search for a maximally consistent set of model and image features as an exponential tree search rather than as a graph search. A given level of the tree pairs one of the image features with each model feature, plus a special model feature called the null face. The null face branch indicates that the image feature doesn't match any model feature. This allows the matching process to handle sensory data that is not due to the object.

The tree is searched depth first for complete paths. A node may be added to a path only if it is consistent with the other nodes on that path. In the case of an inconsistent node, all paths below that in the tree are pruned away from the search space. In order for a node, n_{k+1}, specifying the feature pair (m_{k+1}, i_{k+1}), to be consistent with the other nodes n_j, $j = 1, \ldots, k$ on a path, the distances and angles between m_j and m_{k+1} must be within the specified error bounds of the distance and angles between i_j and i_{k+1} for each $j = 1, \ldots, k$. The allowable distance and angle ranges for each pair of model features are precomputed, so the consistency check is simply a table lookup.

A complete path, from the root to a leaf of the tree, constitutes a consistent set of model and image features. Such a path assigns each image feature to some model feature or to the null face. A path only ensures consistency among all pairs of nodes along the path. Thus the verification stage checks for global consistency by solving for a rigid body transformation that maps the model feature of each node onto its corresponding image feature. The RAF system has been demonstrated on a variety of recognition tasks, and performs well in the presence of noise and occlusion. Like LFF, the distance and angle constraints are not applicable to 3D from 2D tasks.

The HYPER system of Ayache and Faugeras [Ayache86] is a 2D recognition system that similarly structures recognition as a search for consistent sets of model and image features. HYPER models objects as polygons, and forms a linear approximation to the edges in an image. The matching process starts by matching a "privileged" model segment against compatible image segments. A privileged segment is one of the longer segments in a model. A model and image segment are compatible if the angle between the model segment and its neighboring segment along the contour is similar to the angle between the image segment and its neighbor.

A single matching model and image segment contain enough information to solve for the transformation from a model to an image, since the task is two-dimensional. This assumes, however that the model and image segment were both accurately extracted from the image, with no occlusion or edge finding error. An hypothesized transformation is then checked by transforming model segments, starting with those near the initial matching segment, and determining if any model segments match image segments. When matches are found, the estimate of the transformation is refined. Nearby features are used to extend the match because error in estimating the transformation is less of a problem locally than globally.

While it is possible to extend HYPER to 3D from 3D recognition, the extensive use of distance and angle relations would make it difficult to apply to 3D

from 2D tasks. The major limitations of HYPER are the reliance on privileged segments to start the matching process, and the use of a linear approximation to edge contours. The recognizer depends on a given segment being at the correct position and orientation. For smooth contours, however, the positions and orientations of the linear segments approximating a contour can vary substantially, depending on where along the contour the segmentation starts, and on the amount of sensing error.

3D from 2D

Recently, VanHove has extended the tree search method developed in RAF to the problem of recognizing solid objects from their silhouettes [VanHove87], which is a restricted 3D from 2D task. The idea is to take a three-dimensional model and project it into the image at all possible positions and orientations, and use then these projections to determine the allowable distances and angles between features. Thus, rather than having a single range of allowable distances and angles for each pair of model features, there are a set of allowable distance and angle ranges, each of which has a corresponding set of viewpoints for which it is valid.

The tree search proceeds in the same fashion as for RAF, except that the process of checking whether a given node is pairwise consistent with the other nodes on a path includes a check that the viewpoints are consistent. Experimentally, these constraints appear to be powerful enough to substantially prune the search space. Thus far, however, the system has only been tested on synthetic images of isolated objects.

One of the drawbacks of the silhouette recognition method is that the model formation process is complicated (although it is done offline) and the resulting models may be quite large. Another limitation is that the imaged size of an object must be known in order to compare model distances with image distances. Thus an object must be of a known size and at a known distance in order to be recognized.

The SCERPO system [Lowe87] uses the pruned search framework to address the 3D from 2D recognition problem under perspective projection. Similarly to the systems just discussed, SCERPO looks for a subset of the possible model and image feature pairs that is consistent with a single position and orientation of a rigid object. The major difference is that the viewing model includes perspective projection, and thus the consistency check is a relation on the whole set of model and image pairs, rather than a pairwise relation.

The imaging model includes projection, so it is not possible to simply compare model distances and angles against image distances and angles in order to determine consistency, because distances and angles are not preserved. Instead,

it is necessary to check whether a set of model and image points specify a valid position and orientation of the object. Solving for position and orientation under perspective projection requires up to six pairs of model and image points [Fischler81]. Thus in general many more possibilities will be considered than in the previous systems, because no pruning can be done until at least six point pairs are obtained (or four in the case that they are coplanar).

In SCERPO, objects are modeled as polyhedra, and the initial processing of an image forms linear approximations to the intensity edges. Primitive edge segments are grouped together into features such as (nearly) parallel lines and corners (proximate edges). Only model and image features of the same type are paired by the matching algorithm. The use of proximity grouping makes the system sensitive to the presence of neighboring and occluding objects, because a single image "feature" may be formed from parts of two different objects. Such features will not correspond to a correct interpretation of the image.

The use of approximate parallelism for feature formation restricts SCERPO to cases where perspective distortion is not significant, because perspective projection does not preserve parallelism. Thus it would be faster to solve for position and orientation under weak perspective rather than under full perspective, because the grouping mechanism already restricts recognition to this case.

SCERPO's recognition algorithm starts with an initial guess of the position and orientation of an object. Newton-Raphson iteration is used to solve for position and orientation, so the initial guess must be relatively good for the method to converge. In order to provide a reasonable initial guess, SCERPO uses only those groups of image segments that are composed of at least three line segments. This generally leaves only two degrees of freedom, making the number of alternative positions for each initial match tractable. Relying on edge triples makes a strong assumption about the data, however, and leaves the system quite sensitive to noise and occlusion, because local triples of edges may not be found.

SCERPO's recognition algorithm proceeds by using the current estimate of the transformation to find model features with a single possible corresponding image feature. These features are then added to the set of correspondences, and an improved estimate of the transformation is computed using an iterative least squares method (Newton-Raphson iteration). Six pairs of points are generally needed to solve for position and orientation, so to verify a match requires at least seven distinguishable points on the object to be visible in the image.

In this section we have seen several systems that search for the largest consistent sets of model and image features. While the size of this search space is exponential in the number of model and image features, the space is pruned

using relations such as the distance and angle between points. Bounds on the amount of search required are considered in the discussion of recognition as search, in Chapter 3.

2.1.3 Parallel Relaxation

Relaxation is an approximation technique that uses a local function to iteratively approximate a global function. The computation is local, so all the work of one iteration step can be performed in parallel. Relaxation methods have a large number of applications, but in the case of recognition [Rosenfeld76] relaxation is generally used as a suboptimal graph search algorithm.

A graph is constructed with nodes representing pairs of model and image features, and with edges connecting pairs of nodes. The edges are weighted according to the degree to which two nodes specify a consistent match of model and image features. The graph is similar to the one used by LFF [Bolles82], with the addition of weights on the edges. If the weights are all zeroes and ones, with a zero indicating an inconsistent pair and a one indicating a consistent pair, then the graph is equivalent to the one in LFF. Recall that in LFF a maximally consistent set of model and image features corresponds to a maximal clique of the graph. Similarly for relaxation graphs, a highly consistent set corresponds to a subgraph with high weights connecting all the nodes in the subgraph together.

One means of determining weights for the edges in a graph is to use a spring model [Davis79]. The weight corresponds to the inverse of the amount of stretching necessary for the two image features to match the two model features. If the stretching is too large, then no edge connects to two nodes.

The parallel relaxation algorithm works by removing those nodes (or alternatively those edges) for which some local evaluation function is below threshold, and iterating until no node falls below the threhsold. The remaining connected components of the graph are subgraphs with relatively high weights connecting the nodes. These subgraphs correspond to highly consistent sets of model and image feature pairs. Like the maximum clique algorithm, the method only utilizes the pairwise consistency of the features. Unlike the maximum clique method, however, the sets of features are not guaranteed to be maximally consistent. Rather, they are locally good sets, with goodness determined by the evaluation function.

Parallel relaxation can be viewed as an approximation technique to the exponential problem of finding a maximally consistent subset. The question is how well the particular local evaluation function works.

The maximum number of iterations is one per node in the graph, because there is always at least one node removed at each iteration, or $O(n^2)$ for n model and n image features. A given iteration considers each of the $O(n^2)$ nodes, so the overall runtime is $O(n^4)$. This is much better than the exponential time for finding maximal cliques, but the technique is suboptimal.

2.1.4 Searching Transformation Space

Rather than searching for sets of model and image features that are consistent with a transformation, there are several recognition systems that search the space of possible transformations. Any n-tuple of corresponding model and image features can specify a transformation from a model to an image, where n depends on the recognition task. For instance in 2D recognition, two corresponding model and image points are sufficient to solve for position and orientation. One point is used to translate a model, and the second point is used to rotate and scale it.

Each n-tuple of features defines a possible transformation, so the search space is polynomial, in contrast to the exponential space of corresponding features considered above. The majority of recognition systems that search the space of transformations compute all the possible transformations from a model to an image, and use the generalized Hough transform to find clusters of similar transformations. A cluster of similar transformations is unlikely to arise at random, so a large cluster is taken to correspond to an instance of the object in the image. The limitations of this clustering technique for recognition are considered in Chapter 3.

A few systems explicitly search the space of transformations, computing a possible transformation and then verifying that the transformation is correct [Fischler81] [Augusteijn86]. This is the same kind of control structure as used by the ORA system developed in this thesis. Unlike ORA, however, these systems look for large sets of corresponding model and image features, rather than taking using different kinds of information for alignment and verification. In addition, the systems use various heuristics to limit the search space.

Silberberg, Harwood, and Davis [Silberberg86] use the transformation clustering approach in a restricted 3D from 2D recognition task. The camera model is known, and the objects are polyhedra of a known size that are resting on a support plane at a known distance and orientation (slant and tilt) with respect to the camera. Thus there are only three degrees of freedom: two translations and one rotation.

An image is processed by a Sobel operator, and then linear edge segments

are found. The recognizer pairs each corner in the model with each junction of edges in the image. For each such pair, an estimate of the three free transformation parameters is computed. The method of solving for the transformation may produce zero, one, two or an infinity of solutions. If one or two solutions are obtained then they are used as estimates of the transformation, otherwise the pair is discarded.

By using only corners, this recognition algorithm is restricted in its ability to recognize objects in noisy images and with substantial occlusion. In addition, edge detectors are least reliable at corners, because of the change in direction of the gradient. The method does not extend easily to non-polyhedral objects, because in a linear approximation to a smooth curve the locations of the "corners" are relatively unstable.

The estimated transformations are clustered using the generalized Hough transform, where quantized values of the transformation parameters serve as indices into a three-dimensional table. The table entries with the largest number of elements are taken to be the best possible transformations from the model to the image. For each table entry whose count is maximal, the averaged transformation in that cluster is used to project the model features into the image. Then every projected model feature is paired with the best matching image feature, and a revised estimate of the transformation is computed.

The limitations of this clustering technique are twofold. First, true peaks in transformation space tend to be flattened out by the bucketing operation. If there are only a small number of correctly located corners, the "peaks" will not be distinguishable from the incorrect pairings. Second, the method extends poorly to tasks with more degrees of freedom, because each transformation parameter is a dimension of the lookup table, and the size of the table rapidly becomes unwieldy. In this task the number of buckets is about 10^4, but for a six degree of freedom problem it is on the order of 10^9. These problems are considered in detail in Chapter 3, in the section on transformation clustering, where the generalized Hough transform is modeled as an occupancy problem.

Six degree of freedom transformations

Several systems have applied the technique of transformation clustering to the full six degree of freedom 3D from 2D recognition problem. One system [Linainmaa85] uses three connected vertices to define triples of corresponding model and image points. Similarly to the previous system, a vertex is defined to be the intersection of two linear segments in the image. The need to find triples of vertices connected by edges makes the system sensitive to noise and occlusion in the image.

A triple of corresponding model and image points is used to solve for the

position and orientation of an object under perspective projection. There can be up to four distinct solutions to this problem [Fischler81]. For a solid object there is an additional reflective ambiguity about the plane defined by the three model points, yielding a total of eight possible transformations. The method presented for computing the transformations is complicated, and involves solving a quartic equation.

The full six-dimensional transformation space is too large to search, so only the translation component of the transformation is used for clustering. This greatly exacerbates problems with false peaks and missing true peaks in the clusters. The method was only tested on relatively simple images with few features, so limitations of the clustering method did not become apparent.

Another system that uses clustering for 3D from 2D recognition has been developed by Thompson and Mundy [Thompson87]. This system uses the weak perspective approximation to perspective projection. Thus the six parameters of the transformation are 3 rotations, 2 translations and a scale factor.

The six-dimensional transformation space is too large to search for clusters, so Thompson and Mundy collapse the space onto the two rotation parameters about the x and y axes. That is, when searching for clusters they only consider transformations corresponding to large clusters in this two-dimensional rotation space. Then these transformations are filtered by clustering using the remaining four parameters of the transform.

The system uses a feature called the vertex pair. A vertex pair is two vertices where the base vertex specifies a position as well as the orientation of the two edges leaving the vertex, and the second vertex specifies only position. Vertices are defined at the intersection (or almost intersection) of two edges. Edges are linear approximations to the intensity edges in an image. The two vertices of the vertex pair need not be (and generally are not) connected by an edge. Thus the system uses a feature that is relatively robustly detectable in the image. Furthermore, the two vertices can be far apart, providing a relatively stable estimate of position and orientation.

Thompson and Mundy argue that the problem of solving for a transformation involves higher order equations, and would require iterative approximation methods. Therefore, rather than solving for a transformation they use object models that explicitly encode every possible orientation of an object. As will be seen in Chapter 4, however, there is a simple closed form solution to the problem as long as each feature defines three non-colinear points.

A model is formed by positioning an object at all possible orientations, sampled every 5 degrees, and orthographically projecting each vertex pair into the $x - y$ plane. Only the two rotations out of the plane change distances and

angles, so there are $72 \times 72 = 5184$ possible orientations to consider. A table is formed that maps the (quantized) planar location and angle of each vertex pair at each model orientation to the actual vertex pair and the 3-dimensional position. These model tables are relatively large (about 21K words for each vertex pair in the model), and the positional accuracy of recognition is limited to at best 5 degrees.

At recognition time, each vertex pair in an image is used to index into the model table, and the possible model orientations corresponding to that vertex pair are recovered. The quantized values of the x and y rotation parameters are used to cluster similar transformations. Those transformations that are in large clusters are then further discriminated by clustering using z-rotation, and finally are clustered again using translation and scale. Transformations that are in large clusters in this final space are taken to correspond to instances of an object in the image. The initial clustering based on two parameters is intended to reduce the number of different clusters that must be considered. The analysis in Chapter 3, however, indicates that very few possibilities are eliminated by the initial clustering.

Lamdan, Schwartz and Wolfson [Lamdan87] recently developed a clustering method for recognizing rigid planar objects with arbitrary three-dimensional position and orientation. Their algorithm uses triples of points in a model and an image to find an affine transform mapping the model plane to the image. The computation is closely related to a precursor of the method developed in this thesis, which was restricted to recognizing planar objects [Huttenlocher87]. The major difference is that Lamdan et. al. use a clever technique to perform some of the work offline, so the worst case runtime of their algorithm is better (other than for very complicated images).

Lamdan et. al. note that three non-colinear points define an origin, O, and two linearly independent vectors, $\mathbf{e}_1$ and $\mathbf{e}_2$, in terms of which any (coplanar) point, p, can be written as $(x\mathbf{e}_1, y\mathbf{e}_2)$. By definition, a point $p' = A(p)$, where A is an affine transform, will have the same coordinates (x, y) in terms of the new basis O', $\mathbf{e}'_1$, $\mathbf{e}'_2$, obtained by applying A to O, $\mathbf{e}_1$, and $\mathbf{e}_2$. In other words any three (ordered) non-colinear points in a set of coplanar points form an affine basis, which can be used to express the set of points in a fashion that is invariant under affine transformations.

In their system, a model is processed so that each ordered non-colinear triple (affine basis) of the m model points is used to express the coordinates of the other model points. Each transformed coordinate is quantized and used to index into a hash table, where the basis triplet (and the model in the case of multiple models) is recorded. This preprocessing takes $O(m^4)$ time, and is done

offline.

Recognition consists of taking an ordered non-colinear triple of the i image points and using it as the affine basis for the other image points. Each image point is expressed with respect to this basis and used to index into the hash table, in order to retrieve the model affine bases that are stored in that table entry. A tally is kept of how many times each model basis is retrieved from the table. A basis that occurs many times is unlikely to occur at random, and is hence taken to correspond to an instance of the model in the image. A threshold is used to determine whether a given model basis was retrieved sufficiently many times to correspond to an instance of the model. Every triple of image points may need to be considered, so the algorithm has a worst case running time of $O(i^4)$.

In contrast, the algorithm presented in [Huttenlocher87] for recognizing planar objects has a worst case running time of $O(m^4 i^3)$, because each triplet of model and image points is used to compute the transformation from the model to the image, and to verify a transformation each of the m model points is transformed. Thus, when $m^4 > i$ (for all but very complicated images), the preprocessing performed by Lamdan et. al. improves the worst case running time over the one in [Huttenlocher87].

Lamdan et. al. argue that their algorithm has a probability of succeeding of $1 - \epsilon$ in time $O(i)$. This argument, however, is based on there being a constant ratio $d = \frac{k}{i}$, where i is the number of points in the image, and k is the number of points visible on the instance of the object in the image. In other words, the object must take up a constant percentage of the image, regardless of the complexity of the image, which is not a valid assumption in general.

It should be noted that the affine basis hashing technique is limited to planar models, because a two-dimensional affine basis can only be used to transform a coplanar set of points. Thus the preprocessing technique does not have a straightforward extension to the recognition of solid objects.

Explicit Search

A few systems consider the space of possible transformations by explicitly checking each transformation. An n-tuple of matching model and image features is used to compute a transformation, that transformation is used to map each model feature into image coordinates, and the number of model features that match image features is determined. If more than a certain number of model features are accounted for, the match is accepted. Systems in this class generally search for the first transformation that passes the threshold, and use various heuristics to limit the search space.

The RANSAC system [Fischler81] uses small sets of corresponding model

and image points to solve for a transformation, and verifies each transformation by counting the number of transformed model points that lie near image points. The perspective imaging model is used, and it assumed that the camera parameters are known. Thus the system must be calibrated whenever the camera parameters change. Three corresponding points are used to solve for the four possible transformations mapping a planar model onto an image. A closed form solution for the transformation is described, but the method is complex and involves solving a quartic equation. Rather than using the closed form solution, the implementation of RANSAC uses a heuristic method for computing the transformation, suggesting that the closed form method is not robust.

RANSAC solves for the four possible transformations specified by three corresponding model and image points, and then uses each transformation to map the set of (coplanar) model points into image coordinates. If there is an image point within an error ellipse about a transformed model point, then that model point is accounted for by the transformation. When more than some minimum number of model points are accounted for, a transformation is accepted as a correct match. In Chapter 3 it will be seen that this verification technique has an unacceptably high chance of falsely accepting a match, even for moderately complex images. The process terminates in success when a correct match is found, or in failure after a number of iterations based on an estimate of the prior probability of a correct match. The RANSAC technique is well suited to "poisoned point" problems, where there is incorrect data, but where the correct match still accounts for a large percentage of the data. In contrast, for most recognition tasks the correct match is only a very small part of the total number of transformations.

The alignment and comparison method developed in this thesis has the same control structure as the RANSAC matching method. Both methods compute possible transformations and explicitly verify those transformations. The two methods differ in several crucial respects, however. First, the alignment method uses different kinds of information for the two matching stages. Simple features are used for computing possible transformations, but a more complete description is used for verification. Second, the alignment method uses a simple, fast, robust method of computing a transformation from a triple of points. Third, the alignment method applies to complex images where each correct match accounts for only a small amount of the data, because the verification procedure has a low probability of falsely accepting a match. Fourth, the alignment method finds all the good matches of the models to the data.

A recent paper describes another matching method that recovers the three-dimensional position of planar point patterns by considering possible transformations from a model to an image [Augusteijn86]. An orthographic model of

projection is used, so for real imaging situations the distance from the camera to the object must be known, because of the size change that happens with distance. The method matches four (coplanar) angles in a model to corresponding angles in an image in order to recover the orientation of the model plane with respect to the image. An iterative method is used to solve for the orientation. The technique is not guaranteed to converge, however, and it may converge to incorrect values.

Rather than finding corresponding model and image angles and solving for position, the matcher uses a technique that requires less search but makes very strong assumptions about the data. Each model point is connected to the center of gravity of the model points, and the angles between these edges are computed. Thus the method requires that the object be totally visible in the image, and that it be segmented from any other objects. The angles so computed are sorted based on subtended angle, and model and image angles are matched in order of size. Thus any errors in finding edges or angles that changes the order will cause the matcher to fail.

The system was tested on synthetic data, with random noise added to the angle measurements. The largest perturbation, however, was 2.4 degrees of angular error, which is very small compared to errors in edge localization and finding vertices and angles in real images.

2.1.5 Selecting Possible Models

Most recognition systems are designed to match a single model to an image. If there is more than one object of interest, then each model is matched to the image separately. For small numbers of objects, this is tractable on a serial machine. For large object libraries, however, the time required to successively consider each object becomes prohibitive. Thus some researchers have proposed an indexing stage, prior to matching models to the image [Kalvin86] [Ettinger87]. This indexing process selects a small number of object models to be matched to an image, independent of the positions and orientations of the objects with respect to the image.

The idea behind indexing is to use local, viewpoint invariant information to limit the number of models that must be matched to an image. The information must be local, because the combinatorics of forming sets of local features becomes as complex as the recognition problem in general. The information also needs to be relatively viewpoint invariant because the position and orientation of an object are unknown. Thus the major question is whether local, viewpoint invariant cues can be found that are sufficiently powerful to greatly limit the

number of models that must be matched to a given part of an image.

Relatively little work has been done on the indexing problem. One notable exception is the work of Kalvin, Schonberg, Schwartz and Sharir [Kalvin86] on boundary matching. This system uses pieces of a two-dimensional curve to define "footprints", which are a description that is invariant under planar translation and rotation.

To recognize an image, the footprint method segments edges at sharp concavities, and computes an invariant description of each segment. Sharp concavities are used because they often correspond to points where objects overlap. The footprints are used to index into a hash table, and each footprint proposes the model(s) that correspond to a given curve segment. The method is quite fast even with a library of 100 objects. The major restrictions of the method are that the size of the object must be known, and the footprint computation only applies to planar objects. It should be possible to extend the footprint matching to 3D from 3D recognition. Extending the method to 3D from 2D tasks would be difficult, however, because projection does not leave many attributes of a curve invariant under change in orientation.

Biederman [Biederman85] has proposed a model of indexing in which objects are broken down into a library of primitive components. The primitive components in an image are then used to select possible models, and their positions and orientations. The major problem with this proposal is that a small set of parts cannot be used to accurately represent most shapes (see the discussion in section 2.2). For example the wing of an airplane, which is a reasonable "part" of the object, is not well approximated by any of Biederman's vocabulary of parts. Such an approach requires the parts to be easily recognizable and to come from a small set, otherwise recognizing the parts becomes as difficult as the recognition problem in general.

Ettinger [Ettinger87] has recently implemented a recognition system that decomposes objects into subparts for recognition. The problem domain is two-dimensional forms such as traffic signs. Parts are formed using local features to segment edge contours. The features are ends, cranks, and joins similar to those developed in the curvature primal sketch [Asada86], described below. Parts are found at a variety of scales. An initial object library was developed with 13 objects, that were decomposed into a total of 47 parts. The number of alternative hypotheses explored in recognition increases approximately logarithmically with an increasing number of subparts, in the range of 5 to 50 parts.

2.1.6 Summary of Recognition Work

A number of recognition systems have been briefly considered, and some of their limitations have been identified with respect to the 3D from 2D recognition task described in Chapter 1. In particular, the method of determining possible transformations from an object to an image is central to the success of a recognizer. A method that is of high computational complexity, or that may miss correct solutions can greatly limit the performance of a recognizer.

Most recognition systems structure the matching process as a search for transformations that bring a large number of model and image features into correspondence. These transformations are either found by searching for large sets of corresponding model and image features, or by searching for clusters of similar transformations. Systems that search for corresponding sets of model and image features have an inherently exponential search space. Some systems prune this search space using heuristic rules that may miss possible solutions [Roberts65] [Brooks81a]. Other systems use constraints to prune the search space [Bolles82] [Grimson84]. Systems that search for clusters of similar transformations use the generalized Hough transform [Silberberg86] [Thompson87], which is likely to produce false clusters under many conditions.

Recognition systems often rely on having sensory data that is of the same dimensionality as the recognition task, so that model and image quantities can be compared [Bolles82] [Grimson84]. In systems that do allow for projection from the world to the image, the model of projection affects the complexity of the matching process. The use of perspective projection in [Lowe87] yields a matching algorithm of time complexity at least $O(p^6)$, for p pairs of model and image points. In [Fischler81] only three corresponding pairs of points are required, however the solution method is complex and relatively unstable. A number of systems use the simpler weak perspective imaging model. These systems have not, however, solved the problem of computing a three-dimensional transformation directly from corresponding model and image points. Instead, they either use a heuristic approach to recovering a transformation [Brooks81a], store views of an object from all possible angles to approximate the transformation by table lookup [Thompson87], or restrict recognition to planar objects [Augusteijn86] [Cyganski85] [Lamdan87].

Finally, existing recognition systems generally verify a transformation by requiring a large number of model and image features to be brought into correspondence. In complex recognition tasks such as 3D from 2D recognition, or with cluttered scenes, this verification method is often inadequate. For such tasks, systems should use more complete descriptions for verification and look for negative as well as positive evidence of a match.

2.2 Shape Representation

This section considers some of the relevant research on representing shape, and evaluates the extent to which various representations are applicable to the 3D from 2D recognition task addressed in this thesis. A shape representation uses the primitives extracted by low-level vision routines, such as intensity edges, in order to describe an image in terms of shape properties. This description can then be used for high-level visual functions such as recognition and obstacle detection. With the exception of the LFF [Bolles82] and ACRONYM [Brooks81a] systems, however, most object recognition systems use simple features, and do little processing of an image in order to extract shape primitives. At the same time, most shape representation has been aimed at describing objects rather than forming good features for recognition. One of the few shape representations that has been used in recognition systems is the curvature primal sketch [Asada86].

The shape of an object is defined by its surfaces, or alternatively by its volume, which is the dual of the surfaces. A shape representation is a formal way of describing the shape of an object, usually in terms of surface or volume elements. Two important criteria can be identified for shape representations that will be used to find objects at unknown positions and orientations in cluttered natural scenes:

- A shape representation should be *stable* over viewpoints, depending relatively little on the position and orientation of an object.
- A shape representation should be *reliably computable* from an image, varying little with moderate sensor error and partial occlusion.

According to these criteria, the commonly used linear (planar) approximations to smooth curves (surfaces) are not very good representations. For instance, in a linear fit to a circle, the endpoints and orientations of the line segments approximating the circle are not stable with respect to different starting points along the curve. Furthermore, a small amount of sensor noise can cause the linear segments to be lengthened, shortened, inserted, or deleted, thereby substantially changing the description of an object.

The stability and reliability of a shape primitive depends on the amount of information it contains, and on its spatial extent. A shape primitive that is of small spatial extent but contains a lot of information must be based on fine scale shape properties, and thus will tend not to be reliably computable from an image. A shape primitive that is of large spatial extent will not be reliably computable because of sensitivity to partial occlusion. Thus the most reliable

primitives are those which are relatively local, and contain a relatively small amount of information.

Most existing shape representation schemes have the properties of conciseness and completeness. A shape representation can be concise both by having relatively few different shape primitives, and by representing a given object in terms of a small number of these primitives. A representation is complete if it is possible to draw, or render, an object from its representation. For example, generalized cylinders [Nevatia77] and other symmetry representations have these properties.

The recognition method developed in this thesis requires neither concise nor complete representations in order to find possible transformations from a model to an image. Computing a transformation only requires features that define three points, or two points and two orientations. Thus the features can be local and relatively simple. Once an alignment has been computed, a model is transformed into image coordinates and compared with the image. While the representation used for comparison does need to be relatively complete, it does not need to be viewpoint invariant. Thus verification can use a very simple representation such as the edge contours of an object.

Shape representations are traditionally broken down into two categories based on whether the primitives are volumes or edges. Some examples of each of these kinds of representations are considered below.

2.2.1 Volumetric Representations

Volumetric shape primitives have been used in a number of shape representations, both because they are viewpoint independent, and because of their relative conciseness. The most common volumetric primitives are generalized cylinders (or generalized cones) [Nevatia77] [Brooks81a] [Marr78]. A generalized cylinder consists of an axis and a cross-sectional sweeping rule. In many cases the cross section is restricted to be convex, and the sweeping rule may also be limited to a constant or monotonically changing function. One of the major problems with volumetric representations, however, is that they are difficult to extract from images. For instance, the projected shape of a given generalized cylinder can change substantially depending on the angle from which it is viewed. Thus while volumetric primitives are viewpoint independent, their images generally are not.

The ACRONYM system [Brooks81a] extracted two-dimensional ribbons from images, and used a ribbon as evidence of a three-dimensional generalized cylinder positioned and oriented such that it projected onto the ribbon.

Ribbons were composed of a linear axis, and a monotonically changing cross sectional width. Thus the axis and the widths at both ends completely define a ribbon. From a ribbon, it is generally impossible to determine if the axis length corresponds to the length of the generalized cylinder axis, or if there is foreshortening. Thus even though generalized cylinders are relatively large features (and thus potentially sensitive to occlusion and sensor noise), a single cylinder still does not contain enough information to recover position and orientation from an image.

Generalized cylinder representations commonly employ a small number of primitive shapes such as blocks, wedges, cones and cylinders [Brooks81a]. Thus the true shape of an object is crudely quantized by the representation. This quantization often eliminates the important shape characteristics of an object. For example, the object composed of cylinders shown in Figure 6 could be almost any animal with four legs (except perhaps a giraffe).

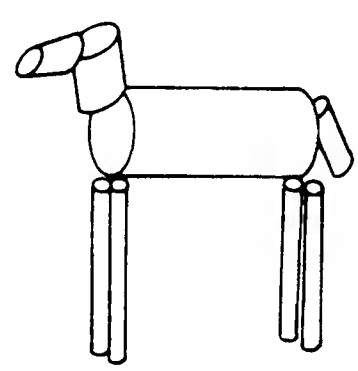

Figure 6. An "animal" composed of cylinders.

One proposal for dealing with the quantization problem has been to use a hierarchy of generalized cylinder representations, with smaller and smaller primitives at each level of the hierarchy [Marr82]. Using finer scale primitives reduces the quantization effect, but it still remains. Certain shapes are not well captured by symmetric volumetric primitives, regardless of the size of the primitives. For example, the contour in Figure 7 is difficult to represent using generalized cylinders. On the other hand, the shape of this contour can be well captured by a multi-scale edge representation such as the curvature primal sketch [Asada86], as discussed in the next section.

There is an interesting historical context to the development of generalized cylinders as a representation. To some extent, generalized cylinders were a reaction against recognition systems that used local viewpoint-dependent descriptions derived from a two-dimensional image (such as the argument against [Barrow76] in [Brooks81b]). In order to use such viewpoint-dependent representations for 3D tasks, many different models of a single object would have been required. Another possibility, however, is to develop edge-based descriptions

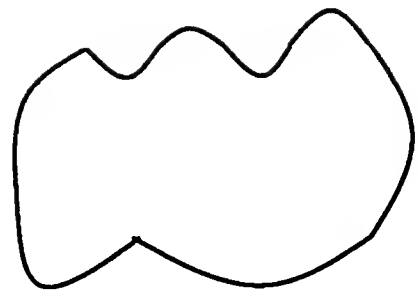

Figure 7. A relatively simple contour that cannot be well represented by a generalized cylinder representation.

that are relatively viewpoint independent. This alternative, which does not require extracting volumetric primitives from a two-dimensional image, is pursued in Chapter 5.

Polyhedra are another commonly used volumetric representation. A polyhedron can be specified by the three-dimensional positions of its vertices [Grimson84] [Bolles82]. Polyhedra are not well suited to recognizing objects with curved surfaces, because for a curved surface the position of the edges and vertices is generally arbitrary. For instance a sphere has many different representations in terms of rectangular planar patches. Thus different instances of the same object may be represented very differently.

Super-quadrics, which have been used for representing objects in computer graphics [Barr81], have also been proposed as a representation for vision systems [Pentland85]. Super-quadrics are like generalized cylinders in that they are analytic specifications of a volume, however they provide a more general set of shapes, characterized by latitude, longitude, and two surface parameters. This makes shape quantization less of a problem than with generalized cylinders, but at the cost of turning the problem of matching two shapes into a search in a large space of possible shapes.

2.2.2 Edge-Based Representations

Unlike volumetric representations, edge-based descriptions can generally be extracted reliably from images, since the primitives are two-dimensional in nature. The utility of an edge-based representation for 3D from 2D recognition is limited, however, by the extent to which the representation is sensitive to changes in viewpoint. For instance, a representation using distances or angles will be very sensitive to slight changes in viewpoint, because these quantities are not preserved under projection.

A number of edge-based representations have been developed, which can be categorized according to whether the primitives are edge-contours [Asada86]

[Hoffman86], or regions of the image (such as symmetry regions) [Brady84] [Fleck85]. As with volumetric representations, these representations have generally been developed to produce concise and complete descriptions of an object.

The codon representation proposed by Hoffman and Richards [Hoffman86] is based on segmenting an object at negative minima of the principal curvature of a surface. The argument for using local negative minima of curvature is based on the fact that these points correspond to concavities, where objects often overlap, or where different parts of an object are connected. The representation does not, however, address the issues of how reliably computable these quantities are, to what extent an image should be smoothed, or how to form multi-scale descriptions.

The curvature primal sketch [Asada86] is a multi-scale description of a planar curve based on "knot points" along the curve, where the knot points correspond to curvature discontinuities. It is the knot points, rather than the shape of the contour between the knot points that captures the shape of an object (e.g., linear segments or circular arcs could be used between knot points).

It is not possible to detect curvature discontinuities in a digitized image unless the digitization is extremely fine compared to the magnitude of changes in the contour. Thus the curvature primal sketch actually identifies points of high curvature and high change in curvature. Magnitude of curvature is not preserved under projection, however, so these points are not necessarily invariant as an object rotates in space, and thus the method does not extend well to 3D from 2D recognition.

In order to locate knot points, a curve is first segmented by matching a small number of analytic functions to the curve. These functions represent simple parts of a curve such as corners, ends, bumps, and cranks. The matching is done based on the smoothed curvature and its derivative computed from the edge contour, at multiple scales of smoothing. Once a curve has been segmented, each segment defines a set of knot points. Segmenting the curve and finding the knot points given the segments are both potential sources of error. Therefore, it would seem better to just compute the knot points directly from the curve, because they are the primary representation of an object. After extracting the knot points, they could be classified according to the kind of curve segment between two points.

The use of analytic functions to represent shape in two-dimensions suffers from the same problem as analytic volumes such as generalized cylinders; the true shape of an object is quantized in terms of the primitive functions. If the functions are relatively simple, such as with constant curvature arcs or generalized cylinders, then this quantization can be quite large. On the other hand,

if the functions are complex then there is a smaller quantization effect, but the functions may be difficult to compute.

Symmetry-based shape descriptions [Brady84] [Fleck85] [Blum78] suffer from the problem that symmetry is a relatively global attribute of an object. Therefore it is difficult to encode detailed shape information in terms of a symmetry representation. Furthermore, the computation of symmetries is quite sensitive to partial occlusion, and to noise in the edge detection process.

The symmetric axis transform (SAT) [Blum78] is a grassfire technique for finding the axes of a region defined by a closed contour. This method suffers from the additional problem that it is highly sensitive to noise in the edge contour. A slight bump or dent in an edge can substantially change an axis. Therefore the method cannot be used to reliably extract axes from real images.

While smooth local symmetries (SLS) [Brady84] are an improvement over representations such as the symmetric axis transform (SAT) in terms of sensitivity to noise, problems with partial occlusion are inherent to symmetry representations, and thus still remain. In addition, the information preserved by symmetry representations, such as lengths and orientations of axes, is not invariant under projection. This makes the representation inappropriate for 3D from 2D recognition tasks.

A recent system for matching planar curves uses a shape representation based on inflection points (points where the curvature changes sign) [Mokhtarian86]. The system forms a scale-space description of a curve in terms of the locations of the zero crossings of curvature at multiple scales. Thus, rather than segmenting a contour, the location of the inflections with respect to the arc length parameter, s, is used as a representation. The major limitation of this representation is that the locations of inflections alone do not capture certain important information about an edge contour, such as straight versus curved.

Schwartz and Sharir [Schwartz87] recently proposed a method for deriving "characteristic curves" from noisy data. Given an edge array, an epsilon neighborhood is defined around each edge pixel. The characteristic curve consists of the shortest path through that epsilon neighborhood. A fast algorithm for performing this restricted shortest path problem is presented. In addition to speed, they argue that the characteristic curve approximation is better than other smoothing techniques, such as convolution with a gaussian.

The characteristic curve method generates a straight line approximation to a curve, because the shortest path between two points is a straight line. Thus like any other linear approximation technique, the characteristic curve method will produce unreliable descriptions of smooth curves, because the ends of the linear segments will depend on noise in the contour and on where along an arc

the segmentation process starts.

2.2.3 Summary of Shape Representation Work

Most existing shape representations have not been developed explicitly for use in recognition. Even those representations that are intended for use in a recognition system do not make a distinction between the information that is important for computing a transformation, and the information that is important for verifying a match.

Shape representations that are used to compute a transformation should be sparse, relatively stable over changes in viewpoint, and reliable with respect to sensor noise and partial occlusion. A sparse description will generally produce fewer matches to an image, thus reducing the amount of search in recognition. Stability and reliability are important for matching partially occluded objects in natural scenes. Many shape representations are not sparse, stable or reliable. For instance linear (or planar) approximations to curves (or curved surfaces) are highly sensitive to noise and partial occlusion. Symmetry representations are sensitive to partial occlusion and are also relatively variable over changes in viewpoint. Volumetric primitives such as generalized cylinders are difficult to extract from two-dimensional images, and are sensitive to partial occlusion.

In contrast, verifying a transformation requires a shape representation that forms a more complete but less abstract description of an object. A complete description is important because accepting a match may involve comparing an entire model to an image. A concrete representation, such as the edge contours of an object, is useful because it can be compared directly against the image edges. This makes it unnecessary to perform a time consuming and potentially error prone extraction of a more abstract description.

Chapter 3

Recognition as Search

Model based recognition is the problem of finding transformations that map models onto their instances in an image. Most recognition systems find potential transformations either by searching for large sets of corresponding model and image features, or by searching for clusters of similar transformations. The correctness of a transformation is then verified by requiring it to map some minimum number of model features onto corresponding image features. Some recognition systems also posit an initial stage of processing to select possible models [Kalvin86], rather than considering the possible transformations of each model.

This chapter formalizes the recognition problem, and then considers each of the three stages of processing: selecting models, hypothesizing transformations, and verifying hypotheses. The next section presents a general formulation of the model based recognition problem, followed by a section on selecting models. Then the two major paradigms for hypothesizing transformations are considered: search for large sets of corresponding model and image features, and search for clusters of transformations. An evaluation of the generalized Hough transform points out its limitations as a transformation clustering technique for recognition. Finally, the verification problem is addressed.

3.1 Problem Formulation

Recognition can be posed as the problem of finding those object models and transformations such that the difference between the transformed model and part of the image is sufficiently small. Given an image, I, a set of models, $\mathcal{M}$, an allowable class of transformations mapping a model to an image, $\mathcal{T}$, and a function d that measures the difference between a transformed model and an image, we seek

$$\{(T, M) | M \in \mathcal{M}, T \in \mathcal{T}, d(T(M), I) < \epsilon\}.$$

The range of allowable models in $\mathcal{M}$, and transformations in $\mathcal{T}$ are both limited by a given recognition system. The types of models are generally restricted by the choice of shape representation. For instance, if objects are modeled as

polyhedra then the allowable set of objects are nearly-polyhedral forms, because objects with smooth surfaces are not well captured by such a representation.

The class of allowable transformations is affected both by the imaging process and by the allowable deformations in the models themselves. In 3D from 2D recognition systems, the transformation from a model to an image includes projection from the world into the image plane. Thus $\mathcal{T}$ encodes the imaging model used to approximate the projection. If general nonrigid deformations of objects are admitted, it is possible to transform very dissimilar shapes into one another, so the degree of deformation in T must be included in the distance measure, yielding,

$$\{(T, M) | M \in \mathcal{M}, T \in \mathcal{T}, d(T(M), I, T) < \epsilon\}.$$

Formulating recognition as a search across all models and transformations does not specify how this search takes place. There are several issues involved in searching the space of possible models and transformations: whether models are selected before considering possible positions and orientations, how possible transformations are found, and how it is determined if a transformation is correct. The remainder of this chapter considers these issues.

3.2 Selecting Models

In most recognition systems, each model, M, in the set of models, $\mathcal{M}$, is matched to an image separately. Since these matches are all independent, they could be performed in parallel on a machine with sufficiently many processors. On a serial machine, however, the comparisons are done one at a time. For a large number of models, linear search of the set of models becomes prohibitively slow. Thus some researchers (e.g., [Kalvin86]) have proposed an indexing stage that selects a small number of candidate models without knowing the transformation from each model to the image. This initial stage generally uses a table lookup or other sublinear time access mechanism.

The initial model selection uses a set of keys, or signatures, $\mathcal{K}$, that are distinctive features of objects. Each model $M \in \mathcal{M}$ has an associated set of keys $K_M \in \mathcal{K}$. Any individual key $k \in K_M$, if found in an image, I, indicates that an instance of the model M may be present in I. The set of models, $\mathcal{M}$, defines a function F that maps a key k onto those models having that key, $F(k) = \{M | M \in \mathcal{M}, k \in K_M\}$. Each key must be local so that it is insensitive to partial occlusion. Furthermore, the keys must be relatively invariant with respect to different viewpoints, because they are used before the transformation from a model to an image is known. Moreover, in order for the indexing operation to

propose only a small number of models, each key must correspond to only a few different objects (i.e., the set $F(k)$ must be small for each k).

An image is processed to extract the set of keys, K_I, present in the image. Then a set of candidate models, $C \subseteq \mathcal{M}$, is formed by using each $k \in K_I$ to hypothesize the models it corresponds to, yielding

$$C = \bigcup_{k \in K_I} F(k).$$

The function F can be implemented as a table lookup, so that the time to map a given key to a set of models is relatively independent of the total number of models.

The footprint matching method developed in [Kalvin86] is an instance of such a model selection process, but is limited to 2D recognition. It is difficult to develop keys that are invariant under projection and insensitive to partial occlusion, yet are sufficiently distinctive to discriminate between different objects. Perhaps this is why effective 3D from 2D indexing methods have not been developed.

3.3 The Space of Possible Corresponding Features

One method of finding possible transformations from a model to an image is to search for those sets of corresponding model and image features where a single transformation maps each model feature onto its image feature. Such a set identifies a group of image features that are consistent with a given model being positioned and oriented according to the transformation. Large sets of consistent feature pairs are unlikely to occur at random, and thus are taken to indicate instances of an object in an image.

A number of recognition systems [Roberts65] [Brooks81a] [Lowe87] [Bolles82] [Grimson84] perform such a search for large sets of corresponding model and image feature pairs. Given a set of image features, I, and a set of model features, M, the idea is to find the sets,

$$P_T = \{(f_m, f_i) | f_m \in M, f_i \in I, d(T(f_m), f_i) < \epsilon\},$$

that maximize $|P_T|$. The function d measures the mismatch between a transformed model feature and an image feature, ϵ is the maximum allowable mismatch, and $T \in \mathcal{T}$.

Let the size of I, $|I| = i$, and $|M| = m$. If each model feature is allowed to match only one image feature, and vice verse, then there are

$$S = \sum_{k=1}^{m} \binom{i}{k} \binom{m}{k} k!$$

possible sets, P, of corresponding features to consider. Assuming that $n = m = i$, a relatively tight lower bound on the size of this expression can be obtained (due to A. Shamir),

$$\Omega\left(\frac{n^n}{\sqrt{2\pi}e^{n-2\sqrt{n}}}\right).$$

The upper bound is similar, being

$$O\left(\frac{n^{n+1}}{\sqrt{2\pi}e^{n-2\sqrt{n}}}\right).$$

To get a lower bound, note that S must be larger than s_l, the largest single term of the summation. Similarly, an upper bound is given by the fact that S is smaller than $n \cdot s_l$. If we denote $S = \sum_k s_k$, where the s_k are the individual terms for each k, then s_l corresponds to the largest k for which $s_k < s_{k+1}$.

If $s_k < s_{k+1}$ then,

$$\frac{n!^2}{k!(n-k)!^2} < \frac{n!^2}{(k+1)!(n-k-1)!^2}$$

which simplifies to,

$$(k+1) < (n-k)^2.$$

These two expression are approximately equal when

$$k \approx n - \sqrt{n}.$$

We actually have to keep k an integer, so we can think of k as $n - [\sqrt{n}]$, where $[x]$ is the integer part of x. From now on $\sqrt{n}$ should be thought of as $[\sqrt{n}]$.

The largest term, s_l, is obtained when $s_k \approx s_{k+1}$, which is when $k = n - \sqrt{n}$, so

$$s_l = \binom{n}{k}^2 k! = \frac{n!^2}{\sqrt{n}!^2(n-\sqrt{n})!}.$$

We now use the fact that

$$(n-\sqrt{n})! \approx \frac{n!}{n^{\sqrt{n}}}.$$

The reason is that

$$n! \approx (n-\sqrt{n})! \cdot n^{\sqrt{n}}$$

because

$$(n-\sqrt{n})! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-\sqrt{n})$$

and

$$n^{\sqrt{n}} \approx (n-\sqrt{n}+1) \cdot (n-\sqrt{n}+2) \cdot \ldots \cdot n$$

(because, for large n, there are $\sqrt{n}$ terms each one approximately equal to n).

So

$$s_l = \frac{n! n^{\sqrt{n}}}{\sqrt{n}!^2},$$

where $\sqrt{n}$ again means $[\sqrt{n}]$. Using Stirling's approximation for $n!$,

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n,$$

$$s_l \approx \frac{n^n}{\sqrt{2\pi}e^{n-2\sqrt{n}}},$$

which is a lower bound on S. Similarly, ns_l is an upper bound, because there are n terms, each smaller than s_l.

Thus there are an exponential number of possible sets of corresponding model and image features for m model features and i image features. It is clearly not feasible to search this entire space, even for moderate values of m and i. In the next section, the major methods for pruning the search space are briefly considered.

3.4 Searching for Corresponding Features

Two main methods have been used to prune the exponential space of possible corresponding model and image features. The first method is to use heuristic search rules, such as only considering certain distinguished pairs of model and image features. Heuristic methods were employed by earlier recognition systems such as Roberts' polyhedra recognizer [Roberts65] and ACRONYM [Brooks81a]. The major problem with these methods is that they may overlook a correct match, in which case recognition will fail.

The second pruning method is to use local relations among features to limit the space of possible corresponding model and image feature pairs. These local relations have the property that they will not eliminate any correct interpretations (within certain error bounds). Therefore, unlike the heuristic rules, this technique will not cause a correct solution to be missed.

In the pruned search framework, the space of all possible corresponding model and image feature pairs is considered, subject to local constraints that are used to prune away parts of the search space. This approach is exemplified by the LFF [Bolles82] and RAF [Grimson84] systems. The constraints are n-ary relations between model and image feature pairs. For instance, a pair of model and image features (a_m, a_i) is considered iff $R_1(a_m) = R_1(a_i)$ for some unary relation R_1. These unary relations are generally labels indicating the type of a feature. Similarly binary relations such as distances and angles are used, such that two pairs of features (a_m, a_i) and (b_m, b_i) can only be part of the same correspondence if $R_2(a_m, b_m) = R_2(a_i, b_i)$ for some binary relation R_2. It must be possible to compare model relations with image relations, which limits the utility of pruned search methods for tasks where there is projection from the model to the image.

It is also possible to use n-ary relations for $n > 2$, however computing the relation may be more work than the resultant pruning warrants. For instance, if a ternary relation requires comparing all triples of model features against all triples of image features then the amount of work necessary to compute the relation may be more than the pruning effect. Certain n-ary relations are easy to compute, however. For instance, a relation such as "triples of points belonging to the same feature" can be computed locally for each feature, rather than considering all possible triples in an image.

Relations between features have been used in two different formulations of the matching problem. One method explicitly formulates recognition as a search of an exponential tree of corresponding model and image features, and use the n-ary relations to prune this tree, as in RAF [Grimson84]. The other method forms a graph of consistent model and image feature pairs and then find maximum cliques of the graph [Bolles82]. Both of these approaches were considered in Chapter 2.

Bounds on the amount of search performed by the pruned tree search method have recently been shown [Grimson88]. These bounds are for matched dimensionality recognition problems (2D or 3D from 3D), where distance and angle relations between pairs of features are used to prune the search tree. There can be both extraneous sensory data, and missing data due to occlusion or other causes. Objects are modeled as polyhedra, and the features are line segments. For m model features, i image features, and a model where all the model edges are the same length, the amount of search is bounded above by,

$$m(1+\kappa)^i + m(i-c_0)2^{c_0} + m^2(i^2-{c_0}^2)(1+\frac{\kappa^2}{m^2})^{c_0},$$

where κ is a constant that depends on the model and the amount of sensor noise, and c_0 is the number of image features that actually lie on the model. Thus the amount of search is exponential, but is much smaller than the entire space of corresponding features considered in the previous section.

The most powerful local relations for pruning the search for corresponding features involve comparing distances and angles in the sensory data against distances and angles in a model. These constraints do not apply to 3D from 2D recognition tasks, because distances and angles are not preserved under projection.

3.5 The Space of Possible Transformations

A rigid transformation from a model to an image can be computed using a fixed number, t, of corresponding model and image features. The number of

features needed depends on the exact transformation, and on the amount of information contained in each feature. For instance, if the transformation is a two-dimensional translation and rotation, and the features are points, then two corresponding features are needed (one to find the translation, and a second to compute the rotation).

Those t-tuples of features that correspond to a correct match of a model to an image will all specify similar transformations (in the absence of any error they would specify the same transformation). A clusters of similar transformations may thus indicate a correct match of a model to an image. Therefore, rather than searching the exponential space of corresponding model and image features, it is possible to search for clusters in the polynomial space of transformations. A number of recognition systems use such a transformation clustering approach [Silberberg86] [Thompson87] [Lamdan87].

Given that t corresponding model and image points determine a transformation, each t-tuple of model and image point pairs could potentially specify a different position and orientation of an object. The number of correspondences is the number of t-tuples of model points, times the number of t-tuples of image points, times the number of correspondences between t model and image points. So for m model features and i image features, the number of possible distinct transformations is bounded by,

$$\binom{i}{t}\binom{m}{t}t!,$$

which is $O(m^t i^t)$ for some constant t. This is a loose upper bound on the number of transformations, because in general different t-tuples will correspond to the same solution.

In the next chapter, it is shown that $t = 3$ in the case of the weak perspective imaging model, using features that consist of a single point. Thus the maximum number of different positions and orientations is $O(m^3 i^3)$. Under perspective viewing, on the other hand, six corresponding points are required to solve for a unique transformation [Fischler81], so there are $O(m^6 i^6)$ possibilities. For five corresponding points there are at most four possible solutions to the perspective equations [Ganapathy85], yielding $O(m^5 i^5)$ possible transformations.

Even the set of $O(m^3 i^3)$ possible transformations from a model to an image is a relatively large space to search explicitly. This, however, is the number of combinations when each feature specifies only a single point. If the features contain more information, then the number of possible transformations decreases. For instance, if each feature specifies both a position and an orientation, like the features used in LFF [Bolles82] and RAF [Grimson84], then only two corresponding model and image features are needed to determine a transformation

under weak perspective (see Chapter 4). Thus for features consisting of a point and an orientation there are only $O(m^2i^2)$ possible transformations. If a single feature specifies three points (or two points and orientations) then only one pair of model and image features is needed, so there are $O(mi)$ possible distinct transformations.

The space of possible transformations from a model to an image is smaller than the pruned space of possible corresponding features. In the previous section we saw that for features consisting of a point and an orientation, pruned search using distance and angle constraints involves considering up to

$$m(1+\kappa)^i + m(i-c_0)2^{c_0} + m^2(i^2-{c_0}^2)(1+\frac{\kappa^2}{m^2})$$

possibilities for m model features and i image features, where κ is a constant that depends on the model and the amount of sensor noise, and c_0 is the number of image features that actually lie on the model [Grimson88].

In contrast, the maximum possible number of distinct transformations for this matched dimensionality recognition problem is only $O(m^2i^2)$. Two corresponding model and image edges define a transformation as long as the edges are not parallel. The intersection point defines the translation, and the vector in the direction of the edge bisecting the acute angle between the edges defines the rotation, as illustrated in Figure 8.

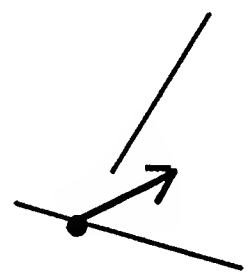

Figure 8. Two intersecting edge fragments define a point and orientation that can be used to solve for a two-dimensional transformation from a model to an image.

The extra work required to search for corresponding features appears to be an inherent property of the method, rather than a consequence of the particular constraints used to prune the search. There are two factors that contribute to extra work. First, searching for a set of corresponding features involves considering all subsets of that set, which accounts for the extra $m(i-c_0)2^{c_0}$ term. Second, the pruning power of the local relations between features depends on the size of the allowable error, so more correspondences are considered when the error ranges are larger, accounting for the $m(1+\kappa)^i$ term.

In contrast, the number of possible transformations is based on the fixed number features needed to compute a transformation, independent of allowable

errors. Thus when searching the space of possible transformations, only the verification of a transformation is affected by error ranges. Due to the fact that the space of possible transformations is smaller than the pruned space of possible correspondences, the ORA system searches transformation space rather than the space of corresponding features.

Most systems that search the space of transformations use the generalized Hough transform to find clusters of similar transformations. The next section analyzes this clustering technique, by modeling it as an occupancy problem. The analysis shows that the technique is of limited utility when there are a large number of possible transformations, such as for 3D from 2D tasks, or when an image is cluttered. Therefore, rather than clustering transformations, the ORA system explicitly verifies a transformation by aligning a model with an image.

3.6 Searching for Transformations

A number of recognition systems solve for the possible transformations between a model and an image, as described in the previous section, and then search for large clusters of similar transformations (e.g., [Thompson87] [Silberberg86]). Some systems use the largest cluster, or largest few clusters, and others look for any clusters above some threshold size.

Clustering in an n-dimensional parameter space is a somewhat difficult problem, however. The two most common techniques are k-means clustering and the generalized Hough transform. These techniques both take as input a set, V, of parameter vectors, or points in an n-dimensional parameter space, and produce a set of subsets of V, where each subset is a cluster of similar parameter vectors.

The k-means method is an iterative technique that starts by dividing the parameter vectors into k groups and computing a centroid, or prototypical value, for each group. Each vector is then moved into the group whose centroid is closest, using some distance metric. The centroids of these new groups are computed, and the process repeats. The iteration terminates when the difference between the centroid values on two successive iterations is below some threshold.

In order to use the k-means clustering algorithm, it must be possible to define a distance metric for comparing any two parameter vectors. In the case of transformations from a model to an image it is difficult to define a distance metric, because the parameter space consists of both translations and rotations, which are not directly comparable. A further limitation of the approach is that the number of clusters, k, is pre-defined. Thus it must be possible to make a

reasonable guess of how many meaningful clusters there are (in this case, how many instances are in an image).

Object recognition systems tend to use the generalized Hough transform for clustering transformations. This technique works by quantizing the parameter space into discrete n-dimensional buckets, yielding an n-dimensional table of buckets. Each parameter vector is entered into a bucket by quantizing its n parameter values and using them as the indices into the table. The quantization tends to map similar parameter vectors into the same bucket, thus clustering vectors together. There are two problems with this method as presented. First, when the parameter space has a high number of dimensions the table may become prohibitively large. Second, similar transformations may be mapped into different buckets. This happens when a parameter value is near a quantization boundary, and is exacerbated by the presence of noise in the parameter values.

In searching for clusters of similar transformations, it is assumed that groups of similar transformations correspond to a correct match of a model to an image. When clustering is implemented using the generalized Hough transform, however, groups of similar transformations may also be likely to occur at random. The likelihood depends on the number of buckets (the coarseness of the quantization), and on the number of transformations entered into the table. Various factors tend to limit the number of buckets, thereby increasing the likelihood of random clusters. For instance, a high dimensional parameter space is too large to search, so generally only a subset of the dimensions are used. Furthermore, sensor error makes it unreasonable to have fine quantization. Thus the likelihood of large peaks occurring at random may no longer be small.

The following subsections analyze the conditions under which the generalized Hough transform is a good transformation clustering technique. First reasonable bounds are determined on the size and number of buckets in a Hough table. Then the bucketing operation is modeled as a classical occupancy problem of r balls (the transformations) in n boxes (the buckets).

3.6.1 Quantization, Bucketing and Transformations

The generalized Hough transform quantizes parameter values in order to assign a parameter vector to a bucket. This does not, however, always cause the most similar vectors be grouped together. As an illustration of the problem, consider a parameter that is quantized into buckets of size 10, and the three parameter values 8, 11 and 16. Even though 11 is closer to 8 than it is to 16, the quantization puts 11 and 8 in different buckets, and 11 and 16 in the same bucket. This problem is exacerbated by sensor noise, because a parameter value that is

near a quantization boundary can be moved into a different bucket by a small amount of noise. Two methods have been used to account for these quantization effects. The first method is to enter each transformation into more than one bucket, based on an estimate of the error in computing the transformation. The second method is to compute clusters of transformations over a set of neighboring buckets, rather than a single bucket.

The first method of accounting for noise and quantization is to put each transformation into more than one bucket. Each transformation is used to define a volume of possible transformations, based on an estimate of the error. The transformation is then entered into each bucket in the table that intersects this volume. While the shape of the volume need not be spherical [Clemens86], a spherical volume will be used here to approximate the actual volume. A reasonable estimate of the number of buckets intersected by an n-dimensional spheroid of radius s buckets, centered in the middle of a bucket, is $(2s + 1)^n$. Thus, for example, with an error range of $d = 1$ bucket and a two-dimensional table, each transformation will be put into about 9 buckets. With an error range of $d = 2$ buckets and a three-dimensional table, each transformation will be put in about 125 buckets.

In the second method, peaks are computed over a neighborhood of buckets rather than a single bucket. The neighborhood is generally of size 3^n, where n is the number of dimensions in the table. This guarantees that any two transformations whose parameters are within one bucket width of one another will be included in the same cluster. Thus it is similar to entering a transformation into each of the 3^n buckets surrounding it, and then just computing peaks over a single bucket.

Existing recognition systems tend to use bucket sizes of about 5 pixels for translation parameters, and 5° for rotation parameters [Thompson87] [Silberberg86]. Thus there are approximately 100 buckets for each translation parameter in a 512 pixel squared image, and 72 buckets for each rotation parameter. With these size buckets, even for two-dimensional recognition, which has two translations and one rotation, there are about 720,000 buckets if all three transformation parameters are used for clustering. For three-dimensional recognition, a full six dimensional parameter table would have about 10^{10} buckets. Such a table is not reasonable to search, for instance it is two orders of magnitude larger than the number of transformations between 10 model features and 1000 image features, where each feature pair defines a transformation. Due to the prohibitive size of such a table, clustering is generally done in a two-dimensional subspace of the parameter space [Thompson87] [Silberberg86].

The next section presents a model of the generalized Hough transform as

a classical occupancy problem. This provides a conservative estimate of the number of peaks of a given size that will occur at random, given a particular number of buckets, n, and transformations, r.

3.6.2 An Occupancy Model of the Hough Transform

Given a set of m model features and i image features, the possible transformations from the model to the image are computed. Each parameter of a given transformation is quantized, and the transformation is entered into the appropriate buckets of an n-dimensional table. Buckets containing a large number of transformations (a peak) are taken to correspond to an instance of the object in the image. Large clusters are either identified by a threshold on the number of transformations in a bucket, or by using the largest few buckets. In either case, the peak size, l, that corresponds to a correct match of a model to an image should be large enough that it is not likely to occur at random. In general l will be at most some fraction of the m model features, corresponding to those features that are matched to image features.

This section models the generalized Hough transform as an occupancy problem, in order to obtain an estimate of the probability that a Hough bucket will have peaks of size l or more at random. This probability should be very small in order for the technique to identify mainly the true instances of an object in an image, rather than random groupings of features.

If the transformations from a model to an image were independent and uniformly randomly distributed over the parameter space, then the probability that a given transformation would fall into a particular bucket would be $\frac{1}{n}$, where n is the number of buckets. Thus the probability that r transformations would fall into some particular set of buckets is n^{-r}. To the extent that transformations are not uniformly distributed they will tend to clump together more than indicated by this model, creating larger peaks. Thus modeling the transformations as uniformly randomly distributed yields a conservative model of the likelihood of false peaks.

Given a distribution of r events into n boxes, one can speak of the occupancy numbers, or the number of events in each cell, denoted by $r_1, \ldots, r_n$, where each $r_i \geq 0$ and $\sum r_i = r$ [Feller68]. If the events are randomly distributed such that each of the n^r placements have the equal probability, n^{-r}, then the probability of a given arrangement with occupancy numbers $r_1, \ldots, r_n$ is

$$p_{r_1,\ldots,r_n} = \frac{r!}{r_1! r_2! \cdots r_n!} n^{-r}.$$

This distribution of events is often termed the classical occupancy problem, or Maxwell-Boltzmann statistics.

For the classical occupancy problem, the probability, p_k, that a given cell contains exactly k events is given by the binomial distribution,

$$p_k = \binom{r}{k} \frac{1}{n^k} \left(1 - \frac{1}{n}\right)^{r-k}.$$

We are interested in the probability that a given cell will contain l or more events at random, which is,

$$p_{\geq l} = 1 - \sum_{k=0}^{l-1} p_k.$$

The expected number of cells in a Hough table that will contain peaks of size at least l is thus given by

$$E_{\geq l} = n p_{\geq l},$$

where n is the number of cells in the table. Ideally, the peaks corresponding to correct matches should be of a sufficient size, l, that $E_{\geq l} < 1$. In other words, ideally the expectation should be that there will be less than one false peak in the table.

For even moderate values of n and r, the computation of p_k becomes unwieldy. For sufficiently large values of n, however, the Poisson approximation to the binomial can be used. The error of this approximation is proportional to n^{-1}, so for tables of the size discussed in the previous subsection (10^4 or larger) the error is relatively small. Using this approximation,

$$p_k \approx \frac{\lambda^k}{k!} e^{-\lambda},$$

where $\lambda = \frac{r}{n}$, the ratio of the number of elements entered into the table to the number of buckets.

In addition to the Maxwell-Boltzmann distribution, another common distribution used in occupancy problems is the Bose-Einstein statistic. This distribution has an experimental basis in particle physics, and assigns an equal probability to each of the occupancy numbers, $r_1, \ldots, r_n$. Under the Bose-Einstein model, for large r and n, the limiting case is the so-called geometric distribution, where

$$p_k \approx \frac{\lambda^k}{(1+\lambda)^{k+1}}.$$

This distribution has a long tail as $k \to \infty$, and thus large clusters have a higher probability than under the Maxwell-Boltzmann distribution. Hence the Bose-Einstein distribution would provide a less conservative model of the likelihood of large peaks occurring at random, and the Maxwell-Boltzmann model is used here.

3.6.3 Evaluating the Generalized Hough Transform

To judge the effectiveness of the generalized Hough transform as a clustering technique, this section applies the occupancy model to some representative problems. Consider the case of matching edge segments to compute the two-dimensional translation and rotation from a model to an image. Each pair of model and image edges defines two possible transformations, because either end of the model edge could correspond to either end of the image edge. Let the model have 10 edges and consider images with 50, 250, or 500 edges. In comparison, a complex image may have well over a thousand linear edge fragments. A two-dimensional Hough table of the translation parameters, with 5 pixel buckets, has a total of $n = 10,000$ buckets. In order to allow for 5 pixel error, each transformation is entered into a neighborhood of 9 buckets. Thus the three images correspond to entering $r = 9,000$, $r = 45,000$ and $r = 90,000$ transformations into the table, respectively. The corresponding values of λ are 0.9, 4.5, and 9.

In order for there to be an expectation of less than one false peak in the table, $E_{\geq l} < 1$, the peaks due to a correct match must have at least $l = 6$, $l = 14$, or $l = 22$ transformations for $\lambda = .9$, 4.5, and 9, respectively. Thus in the case of an image with 50 edges, the object can be detected with an expectation of less than one false peak in the table, as long as at least 6 out of the 10 model edges are matched to image edges. When there are 250 or 500 image edges, it is not possible to detect the object with an expectation of less than one false peak in the table. For the image with 250 edges, the probability of random peaks of size at least 10, $p_{\geq 10}$, is about 1 percent, so there will be an expectation of 100 false peaks in the table. When there are 500 image edges, $p_{\geq 10} = .29$, so almost every third bucket in the table will have a peak as large as one corresponding to a match that correctly pairs all 10 model and image edges.

Now consider a three-dimensional Hough table with 5 pixel translation buckets and 5° rotation buckets, for a total of $n = 720,000$ buckets. Each transformation will be entered 27 times to allow for 5 pixel and 5° error. The images with 50, 250 and 500 edges now correspond to $r = 27,000$, $r = 135,000$, and $r = 270,000$ transformations in the table, respectively. The corresponding values of λ are 0.04, 0.19, and 0.38. In order for there to be an expectation of less than one false peak in the table, the peaks due to an actual instance must be of size $c = 1$, $c = 5$, and $c = 6$, for these three values of λ. Thus for two-dimensional edge matching using a full three-dimensional Hough table, the technique yields acceptable results. The cost is high, however, because of the large number of buckets that must be searched. There are several orders of magnitude more buckets in the table than there are possible different transfor-

mations between the model and the image (even for 500 edges, there are only 10,000 possible transformations).

For three-dimensional transformations the size of the Hough table becomes prohibitive, so most recognition systems use only a subset of the transformation parameters to form the table [Silberberg86] [Thompson87]. For example, one 3D from 2D recognition system [Thompson87] uses the two parameters of rotation out of the viewing plane for an initial clustering. The Hough buckets are of size 2°, yielding a total of $n = 32,400$ buckets. An error range of 15° is allowed, so each transformation is entered into an average of $8^2 = 64$ buckets. A model has about $m = 5$ features, and an image has about $i = 3000$ features, resulting in about 20,000 transformations. Thus a total of about $r = 1,280,000$ transformations are entered into the table, for a λ of about 40. In order for the expected number of false peaks in the table, $E_{\geq l}$, to be less than one, the peak size, l, must be 68. This is an order of magnitude larger than the number of model features, m. Peaks of size at least m, which is 5, will occur at random with a probability of 99%. In other words, this initial clustering eliminates virtually none of the candidates.

Following the initial clustering, a secondary clustering is performed using the third rotation parameter. This parameter is again quantized in 2° buckets, so the table consisting of all three rotation parameters has a total of $n = 5,832,000$ buckets. Each transformation is entered in $8^3 = 512$ buckets in order to allow for 15° errors. Thus 20,000 transformations yields $r = 10,240,000$ table entries, and $\lambda = 1.8$. In order for $E_{\geq l} < 1$, the peak size, l, must be at least 11, which is a factor of two larger than the number of model features. Peaks of size 5 occur with a probability of over 1%, yielding an expectation of about a hundred thousand false peaks in the table. Thus the remaining three translation parameters must perform a good deal of work to eliminate the false matches. Even with the full 6 parameters, false matches may easily remain [Thompson87], and further verification is required. Finally, the amount of search required is very large, as several million buckets must be considered in order to find the buckets with peaks.

To get a more complete picture of the utility of the generalized Hough transform for transformation clustering, Table 2 shows how large the peak from a correct match must be for various problems. Each row of the table corresponds to a particular value of $\lambda = \frac{r}{n}$, the ratio of number of table entries to number of buckets. Each column of the table corresponds to a particular probability of a peak occurring at random. To interpret the probabilities, recall that in order for the expected number of false matches to be less than 1, the probability should be less than $\frac{1}{n}$. For example, for a table with 100,000 entries and a λ of 2, the peaks from a correct match must consist of at least 11 transformations before

there will be an average of less than 1 false peak per table. For peaks of size 6 and 7 there will be about 1200 and 300 false peaks, respectively.

$p_{\geq l}$	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
$\lambda = .25$	2	3	4	5	6	7
.5	3	4	5	6	7	8
1	4	5	6	8	9	10
2	6	8	9	10	12	13
4	9	11	13	15	17	18
8	15	18	20	23	25	27
16	26	30	33	36	38	41
32	46	51	55	59	62	65

Table 2. Peak size, l, for different values of $\lambda = \frac{r}{n}$, and different probabilities, $P_{\geq l}$, of peaks at least as large as l occurring at random.

It may seem somewhat surprising that the expected performance of the generalized Hough transform is so poor for clustering problems similar to those in several recognition systems [Thompson87] [Silberberg86] [Linainmaa85]. Recall, however, that the operation was originally used to separate outliers from good data. Its first use in recognition was for relatively simple tasks, where the data corresponding to the correct solution is a fairly large percentage of all the data. In contrast, for recognition in complex scenes the good data is a small fraction of the incorrect data, or "outliers". It just turns out that the method does not scale very well to tasks where the amount of correct data is relatively small compared to the amount of incorrect data.

3.7 Verification

Once a potential position and orientation of a model have been computed, it must be determined whether or not there actually is an instance of the object in the image at the specified location. This verification process generally involves determining how many of the transformed model features are mapped onto image features, within some allowable error range. If more than a certain number of the model features are matched, then an instance of the object is judged to be in the image at the specified position and orientation.

The features used in matching are generally points or line segments, because they can be easily and reliably extracted from images. Extracted points usu-

ally correspond to corners, vertices, or inflections in edge contours, whereas line segments are linear approximations to edge contours. The features are sometimes labeled using local grouping operations, or based on the shape of an edge contour [Bolles82] [Lowe87]. Many systems, however, do not label the features because such classification can be highly prone to error given the levels of noise and occlusion in natural images [Thompson87] [Grimson84] [Lamdan87].

For a set of feature points with no labels, the verification procedure is to take each model point, m_k, in the set of model points, M, and transform it according to an hypothesized transformation, T. Then for each such point, $T(m_k)$, it is determined whether there is an image point, i_j, in the set of image points, I, such that the distance $|T(m_k) - i_j| < \delta$, where δ is the amount of error allowed. If the number of points in M satisfying this check is greater than some threshold, then the match is accepted (e.g., [Fischler81] [Lamdan87]).

While this procedure is adequate for certain tasks, it is not sufficient for images that contain many features. In such tasks, a reasonable error bound, δ, yields a relatively high chance of falsely accepting incorrect matches of a model to an image.

With an error range of δ pixels, a transformed model feature point, $T(m_k)$, will match any image feature point, i_j, that is within a circle of radius δ around $T(m_k)$. If the probability of there being an image feature point at any given pixel is p_f, then the probability of there being some feature point in the circle is the probability of at least one success in n Bernoulli trials, which is $1-q^n$, where $q = 1 - p$ is the probability of a failure on a single trial. The number of trials, n, is the number of pixels inside the circle of radius δ, which is approximately $[\pi\delta^2]$, where $[x]$ is the integer part of x. Thus the probability that an arbitrary $T(m_k)$ will match some i_j is,

$$p_m = 1 - (1 - p_f)^{[\pi\delta^2]}.$$

In order to obtain an estimate of p_f, consider an image containing a few hundred feature points, which is not uncommon in a 512×512 pixel image of a complex scene (for instance see the examples in Chapter 6). The feature points tend to be bimodally distributed, with points more dense where there are objects present, and less dense in the background. It is the more dense number that is relevant here, because hypotheses to be verified will be in these regions, not in the background regions where there are no features to form hypotheses in the first place.

If several hundred feature points were uniformly distributed across an image, then the probability of a feature point at any given pixel would be about 0.002 (500 features out of 250,000 pixels). Therefore in the more dense regions, assume that a reasonable probability of a feature point being at any given pixel

is $p_f = 0.005$. For this value of p_f, and an allowable distance error of $\delta = 5$ pixels, the probability that a model feature taken at random will match the image, as given above, is $p_m = 0.325$. Thus, for images that have moderately dense image features, a point-based verifier with reasonable error bounds has a substantial chance of matching a model point to an image at random.

A transformation will be accepted as a correct match if at least k of the m model features are matched to image features. In the event that k or more features are matched to an image at random, then a match will be falsely accepted. Therefore, given a probability, p_m, of matching a model point to an image point at random, we are interested in the probability that k or more of the m points will be matched to an image at random. The probability that exactly j of the points will be matched, where p_m is the probability of a single match, is the probability of j success in m Bernoulli trials,

$$p_j = \binom{m}{j} {p_m}^j {q_m}^{m-j},$$

where $q_m = 1 - p_m$. Thus the probability that at least k of m points will match, resulting in an error, is

$$p_e = \sum_{j=k}^{m} p_j.$$

The probability of a false match should be very low, because there are many thousands of potential matches between a model and a cluttered image. A $p_e < .0001$ will yield an expectation of less than one false match for most tasks. In the case of a cube, where the seven visible vertices are used as feature points, when $p_m = 0.325$, the probability that all seven vertices will be matched to image points at random is .0004, which is slightly above the acceptable level. The probability that six of seven vertices will match is .006, for five of seven it is .041, and for four of seven it is .161. To allow for partial occlusion and sensor error, only 4 or 5 out the seven vertices can be expected to have been matched. This results in much too high a chance of falsely accepting a match of a model to a random collection of image features. Thus features consisting of just a location are not sufficient to allow a reasonable determination of whether or not a model of seven features is present in even a moderately cluttered image.

If an orientation measure is added to each point, so that verification consists of matching oriented model points to oriented image points, then the probability of a false match enters the acceptable range. This is the kind of verification technique used by the LFF [Bolles82] and RAF [Grimson84] systems when they solve for a globally consistent position and orientation.

Consider an allowable orientation error of $\frac{\pi}{10}$, so that the probability of a given orientation is 0.1. If the probability of a point being at a given image pixel

is 0.005, as before, then the chance of a point at a given orientation being present is the joint probability of these events, so $p_f = 0.0005$. The corresponding probability of a model feature matching an image feature at random is $p_m = 0.0385$. For a model with seven features, as long as at least 4 of the seven features are visible in a correct match, then the probability of a false match will be less than one in ten thousand. If there are only 5 model features, however, the probability that 3 of 5 features will match at random again becomes several in ten thousand.

Thus we have seen that simple verification techniques are not adequate for even moderately complex images. Simply requiring a number of transformed model features to lie near some image feature can result in a relatively high chance of a false match. As was true with the generalized Hough transform, techniques that may be sufficient for simple recognition tasks do not necessarily scale up well to more complex tasks, where most of the sensory data is not due to the object being recognized.

3.8 Chapter Summary

This chapter has formalized the model based recognition task and analyzed certain approaches to the problem. The major issues in recognition are how to select models that might match an image, how to determine possible transformations mapping a model onto an image, and how to verify the correctness of those transformations.

It is difficult to determine what models might match an image without also solving for the transformations that map each model onto the image. Local "signatures" of an object must be found that are both relatively stable across viewpoints and fairly unique. For 2D recognition, some effective edge-based features have been found (e.g., footprints [Kalvin86]), but for 3D from 2D recognition it is an open problem.

Determining possible transformations from a model to an image is generally viewed as a search problem. The two common search techniques are pruned search for large sets of corresponding model and image features, and search for clusters of similar transformations. The pruned space of possible correspondences is larger than the space of possible transformations. This appears to be due to inherent differences in the search methods, rather than to the particular set of relations used for pruning the search. First, finding a large set of corresponding features involves considering subsets of that set. Second, the pruning power of the local relations depends on the expected amount of sensor error. In contrast, a possible transformation is computed from a fixed number

of features. A further limitation of the pruned search method is that the most powerful constraints are not directly applicable to 3D from 2D tasks.

Systems that search for clusters of similar transformations usually use the generalized Hough transform, where quantized values of the transformation parameters serve as indices into a multi-dimensional table. The major limitation of this technique is that there is a substantial likelihood of false peaks for reasonable table sizes and even moderately complex images. To make the probability of a false peak low requires tables that are very large, which would result in a large search space for finding clusters.

Most recognition systems verify a transformation by requiring a number of model features to be brought into correspondence with image features. If each features specifies only a location, then for models with about half a dozen features there is a substantial chance of finding a match at random. Therefore more involved verification procedures are important.

The investigations in this chapter have motivated the particular matching method employed by the ORA system. The space of possible transformations is smaller than the space of corresponding features, so ORA searches the space of transformations. Clustering is a difficult problem, and searching for clusters of similar transformations can yield a number of false matches. Thus ORA verifies each transformation rather than clustering transformations. In order to make the probability of falsely matches low, ORA uses a verification procedure that compares an entire aligned model contour with an image. In the interest of efficiency, the verification procedure is hierarchical, first matching a few local features and then comparing the entire contours if the initial check is passed.

Chapter 4

The Alignment Method

We have seen that a central problem in recognition is the ability to accurately and efficiently determine possible transformations from a model to an image. In this chapter, it is shown that three pairs of corresponding model and image points specify a unique (up to a reflection) position and orientation of a model with respect to an image. In the case of a planar model, the alignment transform is equivalent to an affine transform from the model plane to the image plane. Unlike the two-dimensional affine transform, however, the alignment transform applies to solid models.

A closed form solution for computing the alignment transform directly from three pairs of corresponding model and image points is also presented. The method only involves solving a second order system of two equations, and is thus fast and relatively robust with respect to noise. The method can also be applied to two pairs of oriented model and image points, or to three pairs of model and image edge fragments (without knowing the endpoints of the edges).

4.1 The 2D Affine Transform

Some systems for 3D from 2D recognition of planar objects use a two-dimensional affine transform to map the object plane to the image plane [Cyganski85] [Lamdan87]. A two-dimensional affine transform can be represented as a non-singular 2×2 matrix, $\mathbf{L}$, and a two-dimensional vector, $\mathbf{b}$, such that $\mathbf{x}' = \mathbf{L}\mathbf{x} + \mathbf{b}$, for any two-dimensional vector $\mathbf{x}$. The affine transformation captures the scaling and shearing of a plane under orthographic projection plus scale.

A two-dimensional affine transform can be used to map a plane in space onto its image under orthographic projection plus scale [Lamdan87]. This follows straightforwardly from the definitions of affine and similarity transforms [Efimov80] [Yale68]. The relation between a plane in space and its image under orthographic projection plus scale is the projection of a three-dimensional similarity transform. A similarity transform is trivially affine, because it is a restricted case of the affine transform. An m-dimensional subspace of an n-dimensional affine space is itself affine in m-dimensions. Thus the orthographic projection of a three-dimensional similarity transform is a two-dimensional affine transform.

The converse, that a two-dimensional affine transform corresponds to a position, orientation and scale of a plane in space is not, however, an established result. Therefore, approaches that solve for an affine transform from a planar model to an image [Cyganski85] [Lamdan87] have not established that a solution must correspond to a particular three-dimensional position and orientation of the model plane.

In the next section it is shown that a two-dimensional affine transform always defines a three-dimensional similarity transform that is unique up to a reflection. For a planar model this reflection is not detectable, and thus an affine transform from a model plane to the image plane always specifies a unique three-dimensional position and orientation of the model plane with respect to the image plane.

A restricted case of the two-dimensional affine transform has been used to recover three-dimensional shape properties from skewed symmetries. Kanade and Kender [Kanade83] use an affine transformation between two planar patches to derive a constraint on the relative orientation of the patches. They assume that if there is a two-dimensional affine transformation $\mathbf{x}_1 = \mathbf{L}\mathbf{x}_2 + \mathbf{b}$, for $\mathbf{x}_2$ on P_2 and $\mathbf{x}_1$ on P_1, then P_1 and P_2 are projections of planar patterns P_1' and P_2' in three-dimensions, which are related by a similarity transformation (by a three-dimensional rotation, translation and scale). They then proceed to show that this assumption constrains the relative three-dimensional orientations of P_1' and P_2' to two possible configurations, which are reflections of one another. A restricted form of this constraint is used for recovering three-dimensional shape from skewed symmetries in a two-dimensional image.

Kanade and Kender express their assumption as

$$\mathbf{L}\mathbf{T}_2 = \mathbf{T}_1\sigma\mathbf{R},$$

where $\mathbf{T}_i$ rotates P_i' about the x and y axes so that it is parallel to the $z = 0$ plane, $\mathbf{R}$ is rotation about the z-axis (the viewing axis), and σ is a linear scale factor. From this equation, they derive two equations relating the slant and tilt of P_1' and P_2'. These equations are claimed to always have two symmetric solutions. The proof, however, relies on $\det(\mathbf{L}) > 0$, which is not the case when there is a reflection involved in the transformation from P_2' to P_1'. Furthermore, the uniqueness of the rotation about the z-axis is not established. Thus they do not show that in general a two-dimensional affine transform is always the orthographic projection of a unique three-dimensional similarity transform.

4.2 The Alignment Transformation Exists and is Unique

The major result of this section is that the correspondence of three non-colinear

points is sufficient to determine the position, three-dimensional orientation, and scale of a rigid solid object with respect to a two-dimensional image. The result is shown in several stages. First it is noted that an affine transformation of the plane is uniquely defined by three pairs of non-colinear points. Then it is shown that a linear transformation of the plane uniquely defines a similarity transformation of space, specifying the orientation of one plane with respect to another, up to a reflection.

These two results are combined to show that three pairs of non-colinear points define the position and orientation of one plane with respect to another, up to a reflection. For a planar model, the reflection is not detectable. For a solid model, there are two possible transformations mapping the model to the image. This ambiguity can be resolved using a point not coplanar with the three alignment points. The methods reported in [Ullman87] and [Huttenlocher87] are precursors to the result established here.

Lemma 1. *Given three non-colinear points* $\mathbf{a}_m$, $\mathbf{b}_m$, *and* $\mathbf{c}_m$ *in the plane, and three corresponding points* $\mathbf{a}_i$, $\mathbf{b}_i$, *and* $\mathbf{c}_i$ *in the plane, there exists a unique affine transformation,* $A(\mathbf{x}) = \mathbf{L}\mathbf{x} + \mathbf{b}$, *for any two-dimensional vector* $\mathbf{x}$, *where* $\mathbf{L}$ *is a linear transformation and* $\mathbf{b}$ *is a translation, such that* $A(\mathbf{a}_m) = \mathbf{a}_i$, $A(\mathbf{b}_m) = \mathbf{b}_i$, *and* $A(\mathbf{c}_m) = \mathbf{c}_i$.

This fact is well known in higher geometry [Efimov80] [Yale68], and it can be established easily.

By definition, an affine transformation of the plane is given by

$$x' = a_1 x + b_1 y + c_1$$
$$y' = a_2 x + b_2 y + c_2,$$

for any point (x, y). Denote $\mathbf{a}_m$ by (x_a, y_a) and $\mathbf{a}_i$ by (x'_a, y'_a), and similarly for the other points. A transforms the three points $\mathbf{a}_m$, $\mathbf{b}_m$, and $\mathbf{c}_m$ into $\mathbf{a}_i$, $\mathbf{b}_i$, and $\mathbf{c}_i$, respectively, so we obtain two sets of three equations in three unknowns,

$$x'_a = a_1 x_a + b_1 y_a + c_1$$
$$x'_b = a_1 x_b + b_1 y_b + c_1$$
$$x'_c = a_1 x_c + b_1 y_c + c_1$$

and

$$y'_a = a_2 x_a + b_2 y_a + c_2$$
$$y'_b = a_2 x_b + b_2 y_b + c_2$$
$$y'_c = a_2 x_c + b_2 y_c + c_2.$$

Each of these sets of equations has a unique solution in the case that

$$\begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix} \neq 0,$$

which by the definition of a line is when the points (x_a, y_a), (x_b, y_b), and (x_c, y_c) are not colinear.

Definition 1. *A transformation,* $\mathbf{T}$*, is a* similarity transform *over a vector space* V *when*

$$\|\mathbf{T}\mathbf{v}_1\| = \|\mathbf{T}\mathbf{v}_2\| \iff \|\mathbf{v}_1\| = \|\mathbf{v}_2\|$$
$$\mathbf{T}\mathbf{v}_1 \cdot \mathbf{T}\mathbf{v}_2 = 0 \iff \mathbf{v}_1 \cdot \mathbf{v}_2 = 0$$

for any $\mathbf{v}_1$, $\mathbf{v}_2$ *in* V.

Theorem 1. *Given a linear transformation of the plane,* $\mathbf{L}$*, there exists a unique (up to a reflection) similarity transform of space,* $\mathbf{U}$*, such that* $\mathbf{L}\mathbf{v} \cong \mathbf{U}\mathbf{v}^*$ *for any two-dimensional vector* $\mathbf{v}$*, where* $\mathbf{v}^* = (x, y, 0)$ *for any* $\mathbf{v} = (x, y)$*, and* $\mathbf{v} \cong \mathbf{w}$ *iff* $\mathbf{v} = (x, y)$ *and* $\mathbf{w} = (x, y, z)$.

The structure of the equivalence between $\mathbf{L}$ and $\mathbf{U}$ stated in the theorem is,

$$\begin{array}{ccc} \mathbf{V}^2 & \xrightarrow{\mathbf{L}} & \mathbf{V}^2 \\ \downarrow{\scriptstyle *} & & \updownarrow{\scriptstyle \cong} \\ \mathbf{V}^3 & \xrightarrow{\mathbf{U}} & \mathbf{V}^3 \end{array}$$

where $\mathbf{V}^2$ and $\mathbf{V}^3$ are two- and three-dimensional vector spaces, respectively. The geometrical interpretation of $\mathbf{U}$ is as a rotation and scale of two basis vectors (defining a plane) so that their image under orthographic projection is the same as applying $\mathbf{L}$ to the basis vectors, as shown in Figure 9.

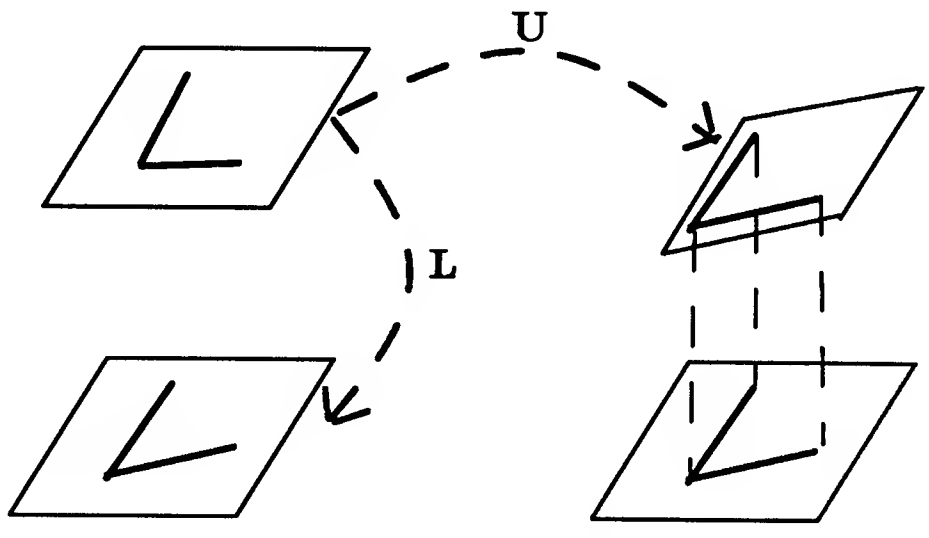

Figure 9. The geometrical interpretation of $\mathbf{U}$.

Proof. Clearly some three-dimensional transformation must exist for any given $\mathbf{L}$, for instance just embed $\mathbf{L}$ in the upper-left part of a 3×3 matrix, with the remaining entries all zero. What must be shown is that there is always a $\mathbf{U}$ that is a similarity transformation, and that this $\mathbf{U}$ is unique up to a reflection.

In order for $\mathbf{U}$ to be a similarity transform, it must satisfy the two properties of Definition 1. We show that these properties are equivalent to two equations in two unknowns, and that these equations always have exactly two solutions differing only in sign. Thus $\mathbf{U}$ always exists and is unique up to a reflection corresponding to the sign ambiguity.

Let $\mathbf{e}_1$ and $\mathbf{e}_2$ be orthonormal vectors in the plane, with

$$\mathbf{e}'_1 = \mathbf{L}\mathbf{e}_1$$
$$\mathbf{e}'_2 = \mathbf{L}\mathbf{e}_2.$$

If $\mathbf{v}_1 = \mathbf{U}\mathbf{e}^*_1$ and $\mathbf{v}_2 = \mathbf{U}\mathbf{e}^*_2$, then by the definition of $\mathbf{U}$ we have,

$$\mathbf{v}_1 = \mathbf{e}'_1 + c_1\mathbf{z},$$
$$\mathbf{v}_2 = \mathbf{e}'_2 + c_2\mathbf{z},$$

where $\mathbf{z} = (0, 0, 1)$, and c_1 and c_2 are constants.

$\mathbf{U}$ is a similarity transformation iff

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = 0$$
$$\|\mathbf{v}_1\| = \|\mathbf{v}_2\|,$$

because $\mathbf{e}^*_1$ and $\mathbf{e}^*_2$ are orthogonal and of the same length.

From

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = 0,$$
$$(\mathbf{e}'_1 + c_1\mathbf{z}) \cdot (\mathbf{e}'_2 + c_2\mathbf{z}) = 0,$$
$$\mathbf{e}'_1 \cdot \mathbf{e}'_2 + c_1c_2 = 0,$$

and hence

$$c_1c_2 = -\mathbf{e}'_1 \cdot \mathbf{e}'_2.$$

The right side of this equality is an observable quantity, because we know $\mathbf{L}$ and can apply it to $\mathbf{e}_1$ and $\mathbf{e}_2$ to obtain $\mathbf{e}'_1$ and $\mathbf{e}'_2$. Call $-\mathbf{e}'_1 \cdot \mathbf{e}'_2$ the constant k_1.

In order for

$$\|\mathbf{v}_1\| = \|\mathbf{v}_2\|,$$

it must also be that

$$\|\mathbf{v}_1\|^2 = \|\mathbf{v}_2\|^2,$$
$$\|\mathbf{e}'_1\|^2 + {c_1}^2 = \|\mathbf{e}'_2\|^2 + {c_2}^2,$$
$${c_1}^2 - {c_2}^2 = \|\mathbf{e}'_2\|^2 - \|\mathbf{e}'_1\|^2.$$

Again the right side of the equality is observable, call it k_2.

It remains to be shown that these two equations,

$$c_1c_2 = k_1$$
$${c_1}^2 - {c_2}^2 = k_2,$$

always have a solution that is unique up to a sign ambiguity. Substituting

$$c_2 = \frac{k_1}{c_1}$$

into the latter equation and rearranging terms we obtain

$$c_1{}^4 - k_2 c_1{}^2 - k_1{}^2 = 0,$$

a quadratic in $c_1{}^2$. Substituting x for $c_1{}^2$ yields

$$x = \frac{1}{2}(k_2 \pm \sqrt{k_2{}^2 + 4k_1{}^2}).$$

We are only interested in positive solutions for x, because $c_1 = \sqrt{x}$. There can only be one positive solution, as $4k_1{}^2 \geq 0$, and thus the quantity inside the square root is $\geq k_2$. Hence there are exactly two real solutions for $c_1 = \pm\sqrt{x}$. For the positive and negative solutions to c_1 there are corresponding solutions of opposite sign to c_2, because $c_1 c_2 = k_1$.

The equation for x does not have a solution when $c_1 = 0$, because the substitution for c_2 is undefined. When $c_1 = 0$, however, $c_2 = \pm\sqrt{-k_2}$, which always has two solutions because $k_2 \leq 0$. Recall

$$\|\mathbf{e}_1'\|^2 + c_1{}^2 = \|\mathbf{e}_2'\|^2 + c_2{}^2,$$

therefore, because $c_1 = 0$,

$$\|e_1'\|^2 \geq \|e_2'\|^2,$$

and hence

$$k_2 = \|e_2'\|^2 - \|e_1'\|^2 \leq 0.$$

Thus there are always exactly two solutions for c_1 and c_2, differing in sign. These equations have a solution iff the similarity transform, $\mathbf{U}$, exists. So there are always exactly two solutions for $\mathbf{U}$ which differ in the sign of the $\mathbf{z}$ component of $\mathbf{x}'$, where $\mathbf{x}' = \mathbf{U}\mathbf{x}$, and hence the sign difference corresponds to a reflective ambiguity in $\mathbf{U}$. ∎

We can now prove Theorem 2, the major result of this section.

Theorem 2. *Given three non-colinear points* $\mathbf{a}_m$, $\mathbf{b}_m$, *and* $\mathbf{c}_m$ *in the plane, and three corresponding points* $\mathbf{a}_i$, $\mathbf{b}_i$, *and* $\mathbf{c}_i$ *in the plane, there exists a unique similarity transformation (up to a reflection),* Q, *such that* $Q(\mathbf{a}_m^*) \cong \mathbf{a}_i$, $Q(\mathbf{b}_m^*) \cong \mathbf{b}_i$, *and* $Q(\mathbf{c}_m^*) \cong \mathbf{c}_i$, *where* $\mathbf{v}^* = (x, y, 0)$ *for any* $\mathbf{v} = (x, y)$, *and* $\mathbf{v} \cong \mathbf{w}$ *iff* $\mathbf{v} = (x, y)$ *and* $\mathbf{w} = (x, y, z)$. *The transformation* Q *is a three-dimensional translation, rotation, and a scale factor.*

Proof. From Lemma 1 there is a unique affine transformation such that $A(\mathbf{a}_m) = \mathbf{a}_i$, $A(\mathbf{b}_m) = \mathbf{b}_i$, and $A(\mathbf{c}_m) = \mathbf{c}_i$. By the definition of an affine transformation, A consists of two components, a translation vector $\mathbf{b}$ and a linear transformation $\mathbf{L}$. We can pick $\mathbf{b} = \mathbf{a}_m - \mathbf{a}_i$, and $\mathbf{L}$ to be the linear transformation such that $\mathbf{L}\mathbf{b}_m - \mathbf{b} = \mathbf{b}_i$ and $\mathbf{L}\mathbf{c}_m - \mathbf{b} = \mathbf{c}_i$, because using $\mathbf{b}_m - \mathbf{b}_i$ or $\mathbf{c}_m - \mathbf{c}_i$ to define $\mathbf{b}$ would produce an equivalent transformation as A is unique.

Given $\mathbf{L}$, by Theorem 1 there is a unique (up to a reflection) rotation and scale, $\mathbf{U}$, such that $\mathbf{U}\mathbf{v}^* \cong \mathbf{L}\mathbf{v}$ for all two-dimensional vectors $\mathbf{v}$. Combining $\mathbf{b}$ and $\mathbf{U}$ specifies a unique similarity transformation Q consisting of translation, rotation, and scale such that $Q(\mathbf{a}_m) \cong \mathbf{a}_i$, $Q(\mathbf{b}_m) \cong \mathbf{b}_i$, and $Q(\mathbf{c}_m) \cong \mathbf{c}_i$. ∎

Note that it is not always necessary to compute the three-dimensional transformation Q. In the event that the model is planar, the affine transformation A is sufficient to map points from the model plane to the image plane.

4.3 Computing the Transformation

The previous section established the existence and uniqueness of the transformation Q mapping model to image points (and the corresponding affine transformation A). This section shows how to compute Q (and A) given three pairs of points (a_m, a_i), (b_m, b_i) and (c_m, c_i), where the image points are in two-dimensional sensor coordinates and the model points are in three-dimensional object coordinates.

Step 0. Rotate and translate the model so that a_m is at the origin $(0,0,0)$, and b_m and c_m are in the $z = 0$ plane. Note that this operation can be performed offline for each triple of model points.

Step 1. Define the translation vector $\mathbf{b} = -a_i$, and translate the model points so that a_i is at the origin, b_i is at $b_i - a_i$ and c_i is at $c_i - a_i$.

Step 2. Solve for the linear transformation

$$\mathbf{L} = \begin{pmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{pmatrix}$$

given by the two pairs of equations in two unknowns

$$l_{11}x_{b_m} + l_{12}y_{b_m} = x_{b_i}$$
$$l_{11}x_{c_m} + l_{12}y_{c_m} = x_{c_i},$$

and

$$l_{21}x_{b_m} + l_{22}y_{b_m} = y_{b_i}$$
$$l_{21}x_{c_m} + l_{22}y_{c_m} = y_{c_i},$$

where $(x_{b_m}, y_{b_m}) = b_m$, $(x_{c_m}, y_{c_m}) = c_m$, $(x_{b_i}, y_{b_i}) = b_i$, and $(x_{c_i}, y_{c_i}) = c_i$. These first two steps yield the affine transformation, A, where $\mathbf{b}$ is the translation vector, and $\mathbf{L}$ is the linear transformation.

Step 3. Solve for c_1 and c_2, using

$$c_1 = \pm\sqrt{\frac{1}{2}(w + \sqrt{w^2 + 4q^2})},$$

and

$$c_2 = \frac{-q}{c_1},$$

where

$$w = {l_{12}}^2 + {l_{22}}^2 - ({l_{11}}^2 + {l_{21}}^2),$$

and

$$q = l_{11}l_{12} + l_{21}l_{22}.$$

Step 4. The scaled rotation matrix is given by,

$$s\mathbf{R} = \begin{pmatrix} l_{11} & l_{12} & (c_2 l_{21} - c_1 l_{22})/s \\ l_{21} & l_{22} & (c_1 l_{12} - c_2 l_{11})/s \\ c_1 & c_2 & (l_{11}l_{22} - l_{21}l_{12})/s \end{pmatrix},$$

where

$$s = \sqrt{{l_{11}}^2 + {l_{21}}^2 + {c_1}^2}.$$

This yields the complete transformation, Q, with translation vector $\mathbf{b}$ and scale and rotation $s\mathbf{R}$. If an explicit representation of the rotation is required, then $\mathbf{R}$ can be derived by dividing all the entries in $s\mathbf{R}$ by s, and then either Euler angles or a unit quaternion can be computed from $\mathbf{R}$.

This method of computing a transformation is relatively fast, because it involves a small number of terms, none of which are more than quadratic. My implementation on a Symbolics 3650 takes about 2.4 milliseconds. The code is in Appendix A.

4.4 Sensitivity to Sensor Noise

This section contains a preliminary investigation into the sensitivity of the alignment transformation to errors in the locations of the image points. Bounds are derived for the error in the l_{ij}, which determines the error transforming those model points that are coplanar with the alignment points. The general problem of model points that are not coplanar is not considered here.

It is assumed that the three model points lie in the $z = 0$ plane, with a_m at the origin, as specified in Step 0 of the previous section. The other two model points are b_m at coordinates (x_{b_m}, y_{b_m}) and c_m at (x_{c_m}, y_{c_m}). The three image points have similarly been translated such that a_i is at the origin, as specified in Step 1, with the other two b_i at (x_{b_i}, y_{b_i}) and c_i at (x_{c_i}, y_{c_i}).

The uncertainty about each image point, b_i and c_i, is a vector of length no longer than ϵ, as illustrated in Figure 10. This circle of uncertainty is approximated by a possible error of $\pm\epsilon$ in the x and y components of b_i and c_i, as shown by the boxes in the figure.

The accuracy with which a transformation can be computed depends on the distance between the origin and b_m and c_m, as well as the angle between b_m and c_m. Longer distances and less acute angles will put the three model points

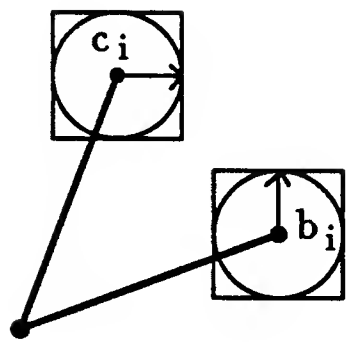

Figure 10. The circle of uncertainty of radius ϵ about the two points b_i and c_i, and its approximation by $\pm\epsilon$ in x and y.

further apart, and thus result in a more accurate computation. Call the angle $\alpha = \angle b_i c_i$, and to simplify the problem somewhat assume that both distances are the same, so $\|b_i\| = \|c_i\| = r$.

The linear transformation matrix, **L** is computed using

$$l_{11} = (x_{c_i} y_{b_m} - x_{b_i} y_{c_m})/d$$
$$l_{12} = -(x_{c_i} x_{b_m} - x_{b_i} x_{c_m})/d$$
$$l_{21} = (y_{c_i} y_{b_m} - y_{b_i} y_{c_m})/d$$
$$l_{22} = -(y_{c_i} x_{b_m} - y_{b_i} x_{c_m})/d,$$

where

$$d = x_{c_m} y_{b_m} - x_{b_m} y_{c_m}$$

.

Using polar coordinates to represent the model points, b_m and c_m yields equations for the l_{ij} in terms of r and α, and a third parameter, θ, the orientation of the vector from the origin to b_m, as illustrated in Figure 11. The parameter θ just depends on the orientation of the model in its coordinate system, and can be assigned an arbitrary value. For example, assume that $\theta = 0$, then the model could be stored so that b_m lies along the x-axis. If $\theta = 0$, then $x_{b_m} = r$, $y_{b_m} = 0$, $x_{c_m} = r\cos\alpha$, $y_{c_m} = r\sin\alpha$. Thus,

$$l_{11} = \frac{x_{b_i}}{r}$$
$$l_{12} = \frac{x_{c_i} - x_{b_i}\cos\alpha}{r\sin\alpha}$$
$$l_{21} = \frac{y_{b_i}}{r}$$
$$l_{22} = \frac{y_{c_i} - y_{b_i}\cos\alpha}{r\sin\alpha}.$$

Perturbing each of x_{b_i}, y_{b_i}, x_{c_i}, y_{c_i} by $\pm\epsilon$ yields an expression for the error of the l_{ij} in terms of the parameters r and α. The error in l_{11} and l_{21} is at most

$$\frac{\epsilon}{r},$$

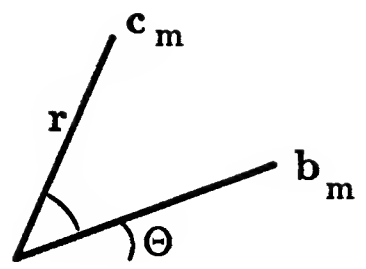

Figure 11. Representing the model points b_m and c_m in terms of polar coordinates.

and the error in l_{12} and l_{22} is at most

$$\frac{\epsilon + \epsilon \cos \alpha}{r \sin \alpha}.$$

Thus in order for the error in the l_{ij} to be at most ϵ, it is required that

$$r \geq \max\left(1, \frac{(1 + \cos \alpha)}{\sin \alpha}\right).$$

This means that as long as $r > 5$ and $\alpha \geq \frac{\pi}{8}$, or $r > 10$ and $\alpha \geq \frac{\pi}{16}$, the error in the l_{ij} will be less than the amount of translational error in b_i and c_i. Most model features will be substantially more than 5 pixels apart, and the angle between the two model vectors will be at least $\frac{\pi}{8}$. As r increases the upper bound on the amount of error in the l_{ij} decreases linearly. For example, if $\alpha = \frac{\pi}{8}$ then for $r = 10$, 20 and 50 the error in l_{12} and l_{22} is at most $.5\epsilon$, $.25\epsilon$, and $.1\epsilon$, respectively.

The transformed coordinates of a model point (x, y) that is coplanar with a_m, b_m and c_m is given by $x' = l_{11}x + l_{12}y$ and $y' = l_{21}x + l_{22}y$. Thus the amount of error in a transformed model point increases linearly the further the model point is from a_m, the origin. The amount of error is at most $\frac{(x+y)}{r}\epsilon$, if the error in the l_{ij} is bounded above by ϵ.

4.5 Alignment Using Oriented Points and Edges

In addition to computing alignments from triples of points, it is possible to use pairs of points and orientations, or triples of edges (with uncertain endpoints) by finding three points defined by these measures.

Suppose we are given two model points a_m and b_m, with three-dimensional unit orientation vectors $\mathbf{a}_m$, and $\mathbf{b}_m$, respectively. Let A_m be the line through a_m in the direction $\mathbf{a}_m$, and B_m be the line through b_m in the direction $\mathbf{b}_m$. If A_m and B_m intersect, then they define a third point for alignment, c_m, as illustrated in Figure 12. Given similar conditions on two oriented image points a_i and b_i, a third image point, c_i can be defined. Now the alignment computation can be performed using three corresponding model and image points.

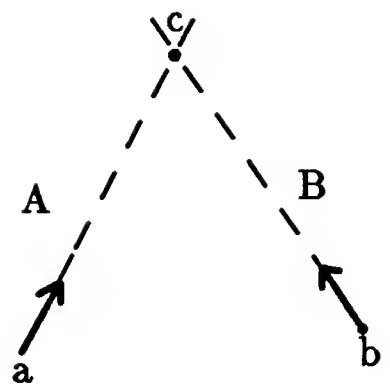

Figure 12. Defining a third point from two oriented points.

The stability of this method depends critically on the distance from the two given points, a_m and b_m, to the intersection point, c_m. If this distance is large, then a small amount of rotational error in either of the two orientation vectors will cause a large positional error in the location of c_m. In real images, the two orientation vectors are generally edge fragments that have a distinguished point at one end, but an uncertain endpoint at the other. A good rule of thumb is to only use intersection points that are within some allowable distance of the endpoints of the actual segments.

Given three model edges $\overline{A}_m$, $\overline{B}_m$ and $\overline{C}_m$, call the lines corresponding to these edges A_m, B_m and C_m, respectively. If A_m, B_m and C_m intersect in three distinct points, then we can define a_m as the intersection of A_m and B_m, b_m as the intersection of B_m and C_m, and c_m as the intersection of A_m and C_m, as illustrated in the first part of Figure 13. Given three corresponding image edges A_i, B_i and C_i, image points can be similarly defined, and the alignment can be computed.

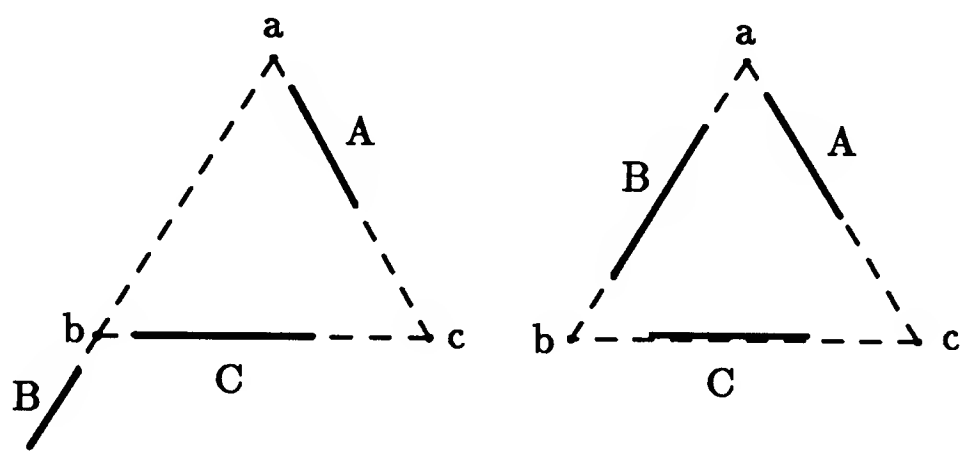

Figure 13. Intersecting segments: i) defining three points from three segments, ii) the segments need not match.

Three segments contain more information than the three induced points used for computing the alignment, so the alignment transformation may not bring

the three model segments into correspondence with the image segments. For instance, the second part of Figure 13, shows a case where the aligned model segments do not correspond to the image segments. Thus it is possible to verify the transformation, by checking that each transformed model edge subsumes part of its corresponding image edge.

In addition to coplanar triples of segments, any pair of segments that define intersecting lines can be used to induce a point at the intersection. A triple of these intersection points will correspond to between three and six edge fragments. Triples of model and image intersection points can be used to solve for the alignment, and the result can then be verified by projecting the three, four, five, or six model edges into the image.

4.6 Chapter Summary

A central problem in recognition is the need to compute possible transformations from a model to an image accurately and efficiently. This chapter has presented a simple, fast, robust method for computing the three-dimensional position and orientation of a solid object with respect to an image. The method only involves solving two sets of linear equations in two unknowns, and a second order system in two unknowns.

In comparison with methods for recovering a transformation under perspective projection, such as those discussed in the first two chapters, the method developed here is fast and stable with respect to sensor noise. A number of other methods have assumed the weak perspective viewing model used here, but they do not solve directly for a three-dimensional transformation from corresponding model and image points.

The method is based on the result that three corresponding model and image points always define a three-dimensional transformation from a model to an image. The result was established by showing that a two-dimensional affine transform defines a three-dimensional similarity transform that is unique up to a reflection. It is well known that three corresponding points define a unique two-dimensional affine transform.

Given the simplicity and stability of the method presented here, the ORA system assumes the weak perspective viewing model, and uses this method for computing possible transformations from a model to an image.

Chapter 5

Representing and Extracting Shape

A model based vision system matches features of stored models against features in an image, in order to determine what objects are present in a scene. A number of kinds of information are useful for defining these features, including shape, texture, color, and motion. This thesis uses two-dimensional shape information derived from edge contours, because edges can be extracted from an image relatively easily.

For recognition tasks like the one addressed in this thesis, where objects are at unknown positions and orientations, may be partially occluded, and can occur in cluttered environments, two important attributes of a shape representation can be identified:

- A shape representation should be *reliably computable* from an image, varying little with moderate sensor error and partial occlusion.
- A shape representation should be *stable* over viewpoints, depending relatively little on the position and orientation of an object.

In order to be reliably computable, a shape feature should not encode detailed shape information, because detailed shapes change under sensor noise. A feature also should not depend on global properties of an object, because these properties change with partial occlusion. In Chapter 2, we saw that most shape representations do not meet these criteria [Blum78] [Brady84] [Lowe85] [Bolles82].

As long as the dimensionality of the sensor data is the same as the dimensionality of the world, a number of shape properties are stable over changes in viewpoint. Very little shape information is preserved under projection, however, so in 3D from 2D tasks most shape features are not stable. For instance, the curvature primal sketch [Asada86] segments edge contours at high curvature points, which are not preserved under projection. Thus the resulting edge segments do not make stable features for 3D from 2D recognition.

In addition to the above two criteria for a shape representation, there are computational reasons to prefer a relatively sparse description of an object. The fewer features a given object has, the fewer possible transformations there are from that object to an image, reducing the amount of search. This requirement trades off against recognizability, because if a representation is too sparse then all the features of an object may be missing from an image. These two opposing

factors suggest that a good representation will encode those parts of an object that are stable across viewpoints and reliably extractable, but not other parts of the object.

This chapter develops an edge-based shape representation that is relatively insensitive to partial occlusion and stable over different viewpoints. Sensitivity to occlusion is minimized by using local edge segments. Stability over viewpoints is obtained by segmenting edge contours at zeroes of curvature, which are preserved under projection. The problem of computing this representation from noisy data is also addressed. Methods are presented for computing orientation, computing curvature, and finding zero crossings of noisy data.

The shape representation developed here addresses the problem of how to find features that are useful for computing possible transformations from a model to an image. As noted in earlier chapters, the best representation for verifying a transformation is quite different. In particular, a representation for verification does not need to be stable across viewpoints because the transformation is known. Furthermore, the representation should be complete rather than sparse, because to check a match may require comparing an entire model with an image.

5.1 Edge-Based Shape

The shape of most objects can be relatively well described using just their edge contours. The major exception is objects that have smoothly changing surfaces. Such objects are not considered in this thesis, but the problem of applying the alignment method to objects with smooth surfaces has recently been addressed in [Basri88]. The major reason for restricting the problem to objects that can be recognized from their edge contours is that edges are relatively straightforward to extract from an image in comparison with other properties of an object, such as surface characteristics.

The edges of an object are usually realized as intensity discontinuities in an image. In order to find discontinuities in a digitized image, however, the digitization must be very fine. Many discontinuities are also accompanied by changes of a relatively large magnitude. Thus edges are generally found by looking for large magnitude changes rather than discontinuities. For example, intensity edges are located at local maxima in the magnitude of the intensity gradient [Marr80] [Canny86].

Many of the intensity edges in an image are not due to edges of objects, but come from other sources such as shadows, texture, and reflections. In addition to extraneous edges, some object edges may be lost because they do not produce a large enough change in the image. The reliability of edges may be improved

by integrating intensity edges with other kinds of discontinuities such as texture and color boundaries [Poggio88]. Even with such information, however, missing and extra edges remain a problem. Given the nature of edge data in real images, a shape representation must be relatively insensitive to spatial noise in the edges, as well as to missing and extraneous data.

The stability of a shape representation across changes in viewpoint is also critical to the success of a recognition system. The most stable properties across viewpoints, however, such as topological properties, do not encode much information about the shape of an object. For example, objects with two holes in them can be anything from a pair of scissors to eyeglasses. In addition, the number of holes in an object is sensitive to partial occlusion, and apparent holes can be produced by accidental juxtaposition of objects.

One means of describing edge contours is by approximation with simple analytic functions such as line segments, cubic splines, or circular arcs. As noted in Chapter 2, however, important shape information may be lost by using these approximations. Therefore the remainder of the investigation in this chapter is concerned with forming edge-based representations by smoothing edge contours at multiple scales, segmenting the smoothed edge contours, and labeling and grouping the segments.

The next section addresses the problem of reliably estimating the orientation and curvature of an edge contour given sensor noise. The following sections consider how to smooth edge contours, how to segment the smoothed contours based on changes in curvature, how to label the resulting segments, and how to integrate multiple scales of segmentation into a hierarchical description.

5.2 Computing Orientation and Curvature

For a curve g, parameterized by its arclength s, the local orientation at a point $g(s)$ along the curve is defined by the unit vector $\mathbf{T_s}$, that is tangent to the curve at that point. There are several ways of estimating the local orientation of a curve. The simplest method is to define a local neighborhood d, and estimate the orientation as $\mathbf{T}_s = g(s-d) - g(s+d)$. The major shortcoming of this method is its rather high sensitivity to noise, because each tangent is estimated using only two points. A method that is less sensitive to noise is to estimate the orientation at $g(s)$ by fitting a line to the points in the neighborhood, $g(s-d), \ldots, g(s+d)$.

A standard line fitting technique is the least squares method, that takes an overdetermined system $\mathbf{Ax} = \mathbf{b}$ and solves for the vector $\mathbf{x}$ that minimizes the error $\|\mathbf{Ax} - \mathbf{b}\|^2$, which is specified by $\mathbf{A}^{\mathrm{T}}\mathbf{Ax} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$. Given a set of points $(x_1, y_1), \ldots, (x_n, y_n)$, the line $y = C + Dx$ which minimizes the error is

determined by

$$\begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum {x_i}^2 \end{bmatrix} \begin{bmatrix} C \\ D \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix}.$$

The best line is $y = \overline{y} + Dx$, where

$$D = \frac{\sum (x_i - \overline{x}) y_i}{\sum (x_i - \overline{x})^2},$$

$\overline{x} = \frac{1}{n} \sum x_i$, and $\overline{y} = \frac{1}{n} \sum y_i$.

This method of computing the best fitting line minimizes only the y component of the squared distance between each point and the best fitting line. Therefore, the method is not appropriate for sets of points where the best fitting line is at an arbitrary orientation. In particular, the best fit to the set of points in Figure 14 would not be useful for computing the tangent vector to a curve.

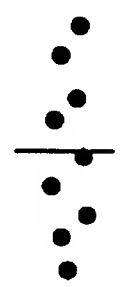

Figure 14. An inappropriate best fitting line using standard least-squares.

This limitation of the standard least squares method has led to the proposal that a linear fit be computed by minimizing the squared distance in the direction normal to the best fitting line, rather than in the y direction [Duda73]. The best fitting line for a set of points must go through the centroid of that point set. Therefore, it is equivalent to translate the set of points to be centered at the origin, and then solve for the best fitting line through the origin as characterized by its unit normal vector, $\mathbf{N}$. This unit vector can be shown to be the $\mathbf{N}$ that minimizes

$$d^2\mathbf{N} = \mathbf{N}^{\mathrm{T}}\mathbf{S}\mathbf{N},$$

where

$$\mathbf{S} = \sum_{i=1}^{n} \mathbf{v}_i {\mathbf{v}_i}^{\mathrm{T}},$$

for the given points $\mathbf{v}_1, \ldots, \mathbf{v}_n$. This form is minimized by taking $\mathbf{N}$ to be the eigenvector of $\mathbf{S}$ that is associated with the smallest eigenvalue. Finding the eigenvalues of $\mathbf{S}$ involves solving a quadratic because $\mathbf{S}$ is a 2×2 matrix. This makes the method somewhat computationally intensive for finding local tangent vectors.

Thus the standard least squares line fitting technique is not applicable to the tangent estimation problem because it only minimizes the y distance, and the method for minimizing the normal distance to the line is computationally expensive to apply at every point along a curve. It is possible, however, to reasonably approximate the normal distance using either the x or y component of the distance, depending on the slope of the best fitting line.

As illustrated in Figure 15a, the normal distance from a point to a line is equally well approximated by either the x or the y component of the distance if the line is $y = x$. In cases where the slope is greater than 1, the x component of the distance is a better approximation to the normal distance than is the y component, as shown in Figure 15b. When the slope is less than 1, the y component is the better approximation, as shown in part c.

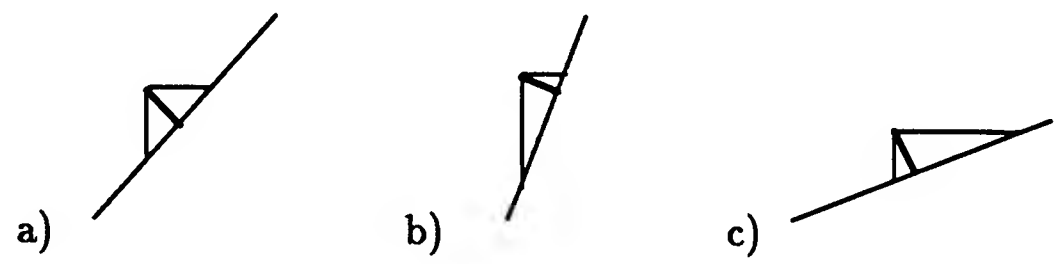

Figure 15. Approximating the normal distance to a line depending on the slope, a) $y = x$, b) using x component, c) using y component.

A simple measure of whether the slope of the best fitting line is greater or less than 1 is the difference in the scatter, or variance, of the x versus y components of the points,

$$s_x = \sum (x_i - \overline{x})^2,$$

where $\overline{x}$ is the mean value of x, and similarly for s_y. If $s_x > s_y$ then the slope of the best fitting line will be less than one, and hence the y distances form the better estimate of the normal distance to the best fitting line. Otherwise the x distances should be used. The best fit can then be computed using the standard technique presented above to minimize the y distance, or its obvious dual to minimize the x distance. The variances s_x and s_y are the denominator in the computation of D in the least squares method, so determining whether to minimize x or y distance also solves part of the final least squares problem.

This slope-based least squares line fitting method applies at all orientations, and is reasonably fast, because the computation is basically the same as for standard least squares. The method is used to compute tangent orientations in the ORA system described in Chapter 6.

Curvature

The curvature, κ, of a path in space $x = g(s)$, parameterized by arc length, s, is given by

$$\kappa \mathbf{N} = \frac{d\mathbf{T}}{ds},$$

where $\mathbf{T}$ is the unit tangent vector and $\mathbf{N}$ is the unit normal vector in the direction in which the path is turning. This formula has a special case for plane curves,

$$\kappa = \frac{d\psi}{ds} = \frac{1}{\rho},$$

where ψ is the angle of the slope of the tangent line, and ρ is the radius of a circle tangent to the curve, called the *osculating circle.*

Rather than computing curvature using the tangent vector, most computational vision algorithms for extracting curvature parameterize a plane curve, $g(s)$ in terms of separate functions of x and y, $x(s)$ and $y(s)$. Then the curvature is given by

$$\kappa = \frac{\ddot{y}\dot{x} - \ddot{x}\dot{y}}{(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}},$$

using Newton's dot notation for the derivatives of x and y with respect to s. This method of computing curvature is highly sensitive to noise, however, because of the computation of second derivative quantities from local image measurements.

Another method that is used to compute curvature is to fit a circle to a small local neighborhood of a curve in order to approximate the osculating circle at that point. Then the radius of the circle, ρ, can be used to determine the curvature. This method is relatively computationally intensive, because it involves fitting a quadratic to a set of points. The method may also require a moderate number of points to get a good estimate of the quadratic. It is less sensitive to noise than the previous technique, however, because it does not involve taking derivatives of noisy image measurements.

In this thesis, curvature is computed from the local tangent vectors to a curve. The tangent vector at each point is computed using a least squares best fit as described above. Angles between neighboring tangents are computed to determine the change in orientation. The change in orientation is then divided by the change in arclength, 1.4 for eight-way connected pixels and 1 for four-way connected pixels. This method has the advantages that it uses a least squares technique to smooth the data, does not compute derivatives of noisy image measurements, and is relatively rapid to compute. In addition, the same method applies to space curves as well as to plane curves.

5.3 Smoothing

One way of forming relatively rich descriptions of an image is to smooth the image at various scales, and then extract the events (such as intensity edges or texture boundaries) that are apparent at each scale. There are several different possible ways to smooth an image and in particular to smooth the edges extracted from an image.

Perhaps the most common method of smoothing is to convolve an image with a two-dimensional lowpass filter, such as a Gaussian, in order to blur out changes in the image [Marr80]. For coarse scales of smoothing, however, there are two problems with such methods. First, peaks in the smoothed image become very broad, and hence localizing edges is somewhat difficult. Second, and more importantly, in images with touching or occluding objects, different objects will be blurred together by the smoothing operation. This is not generally desirable for recognition, where the goal is to separate objects from one another. One way of reducing this problem is to do directional smoothing along the normal to the direction of the gradient (steepest change). This computation is more time consuming than uniform lowpass filtering, and it becomes difficult to perform when the direction of the gradient is changing rapidly compared to the mask size of the filter.

A second method of smoothing is to represent the edge contours in an image in terms of x and y as a function of the arc length, s, along an edge contour, and then smooth the one-dimensional functions $x(s)$ and $y(s)$. If the edge contour is closed, then filtering $x(s)$ and $y(s)$ by convolution also results in a closed curve [Mokhtarian86]. A smoothing filter attenuates the amplitude of the functions $x(s)$ and $y(s)$, however, so the resulting smoothed curve will generally be smaller than the original curve.

Depending on the task, this shrinking of a curve at coarser scales may be problematic. Therefore, other representations such as radius of curvature versus tangent direction have been proposed, which have both the property of preserving closed curves and preserving the size of the smoothed curve [Horn85]. For a multiscale description, however, it is not a problem if the size of a curve shrinks at coarser scales of smoothing. A range of locations along the contour, $[s_k, s_l]$, at one scale will correspond to a single location, s, along the contour at the next coarser scale. This correspondence can be used when moving from one scale to another.

A curve can also be described in terms of orientation versus arclength, $\theta(s)$, where orientation is computed with respect to some reference orientation such as the x axis [Turney85] [Perkins77]. The major problem with smoothing

orientation is that the smoothing must be done modulo the fact that the space of orientations is circular (i.e., an orientation of 2π is the same as 0). Furthermore, the smoothed curve need not be closed even if the original curve is, which may be a problem depending on the task.

Finally, the curvature of a path parameterized by arclength, $\kappa(s)$, can be smoothed. A curve reconstructed from the smoothed curvature need not, however, be closed if the original curve was closed. The smoothing operation attenuates the curvature, so the sum of curvature along the path is reduced, and closed curves can become open. This is only a problem if the curvature is used to reconstruct a smoothed curve. The ORA system described in Chapter 6 uses smoothed curvature to find inflections and peaks in curvature, and then segments the original contour at these distinguished points. For example if there is an inflection at a location s in the curvature smoothed at some scale, σ, then the point $g(s)$ on the curve is marked as an inflection point at that scale of smoothing.

Recently, Saund has argued that any kind of uniform smoothing is inadequate, because different types of smoothing are needed for different parts of an image [Saund88]. He calls for forming a symbolic representation of an image in terms of low-level primitives such as orientation bars, and then smoothing in this space of symbolic primitives. This is very much like Marr's idea of the full primal sketch [Marr82]. The most important issue for this type of smoothing is to demonstrate that the early introduction of symbolic primitives produces a better end representation, because the early categorization of local image information can be quite sensitive to noise.

5.4 Curvature-Based Segmentation

Given a curve that has been smoothed appropriately, the next stage is to segment the curve into "natural" pieces, that are as invariant as possible under changes in viewpoint. A number of shape representations segment curves at high curvature points, because these points are supposedly important in human perception. Studies examining the perceptual importance of maximal curvature points [Attneave54] [Fischler86] do not, however, actually establish that high curvature points are of particular importance in representing shape. Rather, these studies demonstrate that high curvature points contain enough information to capture the shape of a variety of planar contours.

For instance, Attneave's cat, shown in Figure 16a, is constructed by linearly interpolating between the maximal curvature points in a line drawing of a cat. The fact that this linear approximation is still easily recognizable has been used

as evidence that maximal curvature points are of special importance in describing a contour. Lowe [Lowe85], however, points out that the linear approximation in Figure 16b is also easily recognizable as a cat. This approximation is constructed using the points midway between maximal curvature points. Thus a variety of sparse descriptions seem to contain enough information that people can reconstruct a contour. Maximal curvature points, per se, are not necessarily important for forming these descriptions.

Figure 16. Attneave's cat, which is intended to demonstrate the importance of maximal curvature points for representing shape, and Lowe's cat which shows that other points work as well.

It has also been argued that because people choose maximal curvature points in order to describe a contour, these points must be important for representing shape. Studies on which these claims are based (such as [Fischler86]), however, have asked people to segment a planar contour by marking just a few points on the contour. Thus the task calls for a linear approximation, because when connected by line segments the points must suggest the original contour. High curvature points are where a linear approximation is worst, so they are the natural break points for a linear segmentation.

It is possible to segment an edge contour at various points, so the choice of segment boundaries should be motivated by the requirements of the recognition task being addressed. Maximal curvature points are not preserved under three-dimensional rotation and projection, both appearing and disappearing, making them inappropriate for 3D from 2D recognition. For example, in Figure 17 an ellipse is rotated about its minor axis to obtain a circle; illustrating maximal curvature points that disappear. The resulting circle is then rotated around another axis to obtain a different ellipse; illustrating maximal curvature points that appear.

In contrast to maximal curvature points, zeroes of curvature are preserved under projection. A zero of curvature corresponds to a local section of a curve, $g(s)$, for $s = [t-\delta, t+\delta]$, over which the directions of the tangent vectors $\mathbf{T}_s$ are all equal, which is by definition a portion of a straight line. Straight lines are preserved under projection, so the zero curvature of these regions will remain

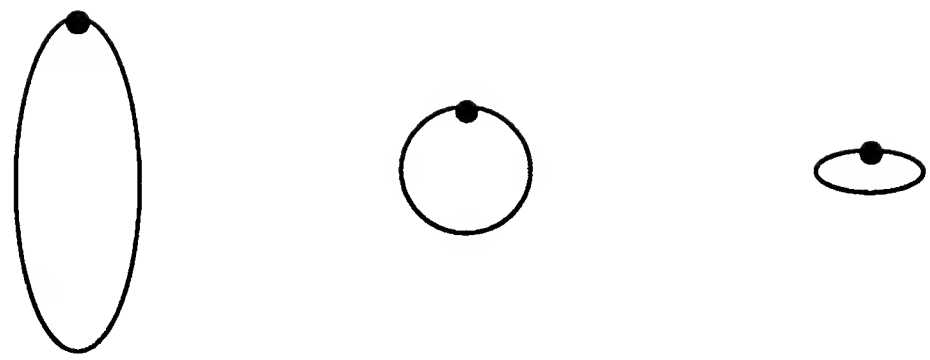

Figure 17. Maximal curvature points are not preserved under projection, both disappearing and appearing with rotation out of the image plane.

zero under projection. A zero of curvature in an image thus corresponds to a zero of curvature on an object, or to an occlusion boundary which can introduce an apparent curvature zero in an image.

5.4.1 Finding Significant Zero Crossings

For an ideal curve, each zero crossing of the curvature will correspond to an inflection point. In noisy data such as edge contours extracted from real images, many zero crossings of curvature will not correspond to inflection points, but rather to the errors in computing the curvature. As in all zero crossing detection problems, some sort of threshold must be defined to separate significant zero crossings from noise.

Several methods have been used to filter the zero crossings of noisy data. One method, illustrated in Figure 18a, is simply to define a minimum height that must be attained by the peak on either side of a zero crossing. Whenever a peak is of sufficient height, the neighboring zero crossings are kept. This thresholding method is very sensitive to noise, however, because it relies on a single point, and that point may be relatively far from the zero crossing.

A second method of filtering zero crossings is to define a minimum slope at the zero crossings, as illustrated in Figure 18b. The slope is computed at each zero crossing, and must be greater than some minimum value for a zero crossing to be kept. This method is less sensitive to noise than the peak method, particularly if the slope is computed using a least squares approximation, because it is based on more data. Furthermore, the data used for thresholding are local to the zero crossing. The major problem with the slope method, however, is that relatively low slope zero crossings can still be significant. For example a zero crossing between two very large but distant peaks may have a low slope. This problem is especially true with curvature data, where a slow change from positive to negative curvature is still important.

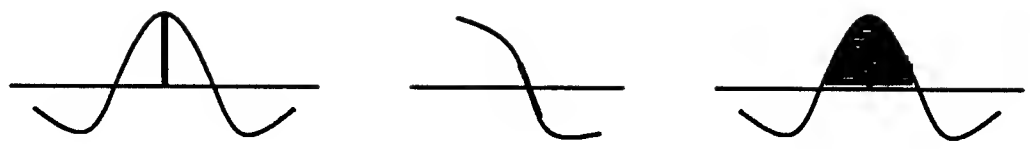

Figure 18. Three methods of filtering zero crossings, a) peak height, b) slope, c) area.

In the ORA system, significant zero crossings are found by defining a minimum area under the curve between two successive zero crossings, as illustrated in Figure 18c. Whenever the magnitude of the area is above a threshold, the neighboring zero crossings are kept. This method does not have the problem of relying on a single point, nor does it have the problem of excluding low slope zero crossings. For curvature data, computing the area between two zero crossings is trivial, because it is the total turning angle over the region, which is just the angle between the tangent orientations to the original curve at those points.

Watt and Morgan [Watt85] have evaluated several different methods of thresholding zero crossings in terms of how sensitive the threshold measures are to noise. They compared peak height, slope and other standard measures against the centroid and mass of a peak, which are both properties based on moments. The moment based properties are more stable under noise, and thus were advocated for use in edge detection.

5.4.2 Segments of a Curve

The methods presented above can be used to find the inflection points of a noisy curve, and these points can then serve as segment boundaries for the contour. The major drawback of using inflection points as segment boundaries is that the straight parts of a contour are not delineated. A side effect of this problem is that a boundary may be located somewhat arbitrarily in the middle of a straight section, as illustrated in Figure 19a.

Figure 19. Segmenting a contour, a) at inflections only, b) at ends of straight segments and inflections.

A simple solution to this problem is to define straight segments as regions of a contour where the curvature is within ϵ of zero for some minimum arclength.

The endpoints of these straight segments are marked as boundaries in addition to those inflections that are not within a straight segments. The resulting segmentation of the same contour is shown in Figure 19b.

Segmenting a contour at inflections and the ends of straight regions thus yields two types of segments: `straight` and `arc`. Each `arc` segment will have either positive or negative curvature. If an `arc` has a sharp local maximum of curvature, then there is probably a discontinuity in the arc. Thus it is further possible to subdivide this class of segments into a `corner` and a true `arc`. The distinction between these two classes is a matter of degree, because under projection an `arc` may appear to be a `corner` and vice versa.

5.5 Hierarchical Edge Description

The previous section presented a method for segmenting an edge contour at zeroes of curvature, yielding straight and curved segments of the contour. By smoothing the curvature, small zero crossings can be removed, resulting in a coarser scale segmentation of the edge contour. For example, the curve in Figure 20a has numerous inflections, corresponding to the zero crossings of κ. Each inflection is marked by a dot. In contrast, for a Gaussian of sufficiently large σ there are only two inflections in $G_\sigma * \kappa$, the convolution of the curvature with the mask, as illustrated in Figure 20b. The inflections are again marked by dots.

Figure 20. Smoothing the curvature removes small zero crossings, and preserves only the larger scale inflection points.

The segmentation of an edge contour using different scales of smoothing of the curvature forms a hierarchical description of the contour. This hierarchical description is a scalespace tree [Witkin85] because under appropriate conditions using a Gaussian filter, coarser scales of smoothing do not introduce zero crossings that were not present at finer scales [Yuille86] [Curtis85]. Thus each segment at a coarser scale will correspond to one or more segments at a finer scale, forming a tree of segments.

Each edge contour in an image will form a separate tree, with a root at the scale at which there are no remaining inflections in the contour. The scalespace

description of a contour can be used to label the segments at a particular scale using the local parent, child, and sibling relations in the tree. This method of classifying segments is more robust than proximity based grouping (such as that performed in [Bolles82] [Lowe87]), because a connected piece of contour is more likely to belong to a single object than are nearby contours.

Figure 21 shows a three-level scale-space curvature segmentation of the edge contours of a widget. Each part of the figure shows the same contour, segmented according to the curvature smoothed at different scales (using Gaussian filters of size $\sigma = 7$, 20, and 40 pixels, respectively). The coarsest scale is at the top of the figure and the finest scale is at the bottom. The endpoints of each segment are delimited by a dot, and straight segments (at that scale) are shown in bold. Each segment is labeled with a letter denoting the level (A for coarsest and C for finest), and a number.

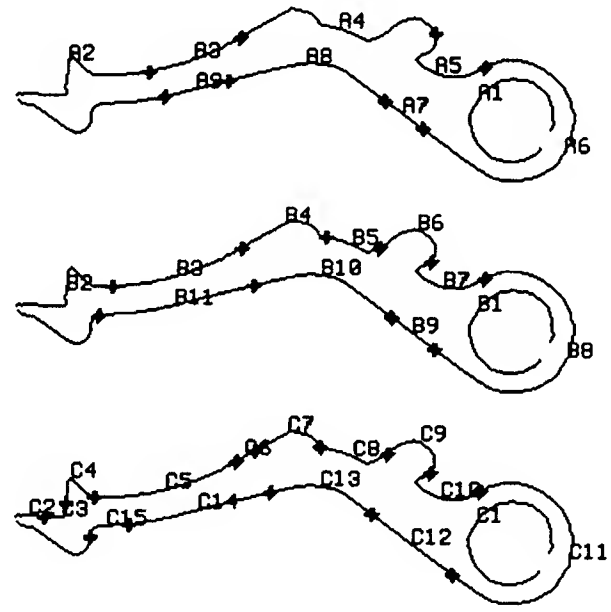

Figure 21. A scale-space segmentation of a widget, where the contours are segmented at inflections in the smoothed curvature. The coarsest scale is at the top.

Recall that each segment of edge contour can be classified according to the three labels `corner`, `arc`, and `straight`. The `corner` and `arc` segments are distinguished by whether or not they contain a high curvature point. The multi-scale description procedure augments these three labels by combining the classifications at multiple scales of smoothing using the segmentation tree.

The multi-scale segmentation in Figure 21 can be viewed as a tree of corresponding segments at neighboring scales of smoothing, as shown in Figure 22. Each segment at a given scale corresponds to one or more segments at the next finer scale. Each segment in the tree is indicated by its label from Figure 21 and by the type of segment: `corner`, `arc`, or `straight`.

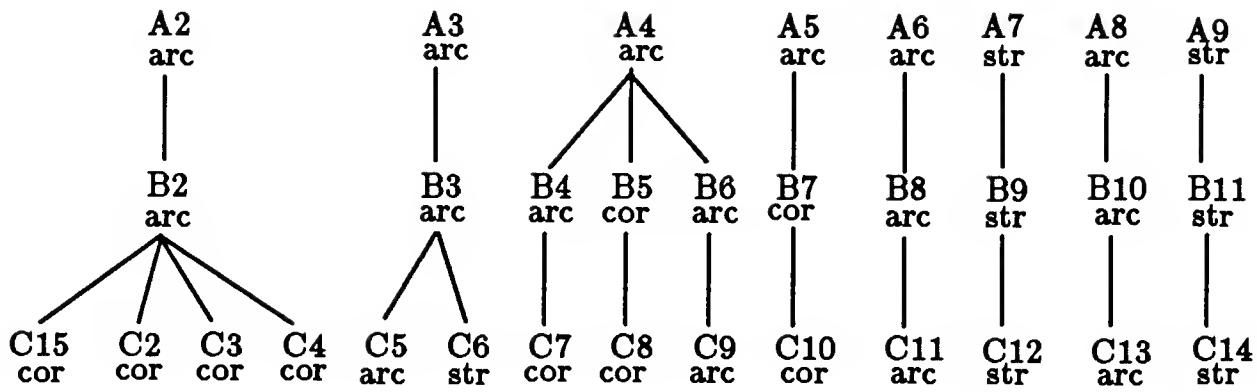

Figure 22. The tree corresponding to the curvature scale-space segmentation in the previous figure.

Multi-scale descriptions of a segment are formed using its ancestors and descendants in the scale-space tree. A segment is classified in terms of its ancestors according to whether it is (primarily) a `single` child, or one of `multiple` siblings. A segment is classified in terms of its descendants according to whether it has (primarily) a `single` child, or `multiple` children.

For example, using this multi-scale description, the `arc` segment `A2` at the coarsest level of the tree is differentiated from the other `arc` segments at the same level, because `A2` is the only segment that is composed of a single `arc` at the next level, and multiple `corner` segments at the finest level.

5.6 Features for Alignment

In order to align an object with an image, three pairs of corresponding model and image points are required. Therefore, features that define three distinguishable points are ideal for determining possible alignments of models with images, because a single feature specifies enough information. Recall from the previous chapter, that two points and two orientation can be used to define a third point, so features that define two distinguishable points and orientations are also sufficient to compute an alignment.

If a feature contains three identifiable points, but the points are not each labeled so that they can be distinguished from one another, then there are several possible alignments. For example, if one point is distinguishable from the other two, then there are two possible alignments, as illustrated in Figure 23. The same is true for two points and two orientations, because the third point that they define cannot be confused with the original two points. If a feature defines three indistinguishable points, then there are six possibilities, corresponding to the different arrangements of three points.

For features that define fewer than three points, more than one model and

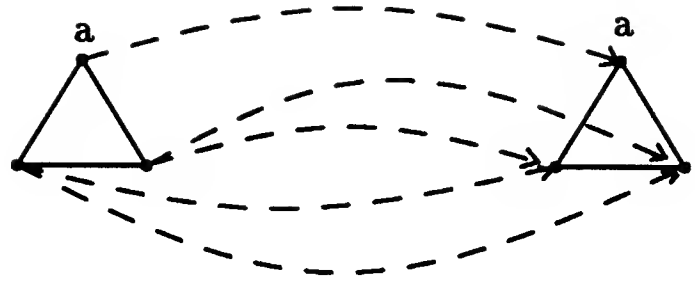

Figure 23. Three points with one distinguished point defines two possible alignments.

image feature are required to compute a possible alignment. If a feature defines two points, or a point and an orientation, then two features are required. If a feature only defines a single point, then three features are needed. The combinatorics of matching with features that define a single point is relatively high, because any triple of model points and corresponding triple of image points could specify a possible alignment. The difficulty of finding features that define at least a point and an orientation is not much higher than the difficulty of finding features that define only a point, so the ORA system described in Chapter 6 uses features that define at least a point and an orientation.

Features that define one point and one orientation are called Class II features, because two features are required to solve for a possible alignment. Those features that define three points, or two points and two orientation, are called Class I features, because only one feature is needed for alignment. Several Class I and Class II features can be defined in terms of a small number of connected edge segments of the kind discussed in the previous section. For example the inflection points at the ends of an `arc` segment each define a point and an orientation, forming a Class I feature. A set of these features are presented in the next chapter, which also discusses how the ORA system extracts features from edge contours.

5.7 Region-based Shape

The spatial arrangement of edges and other primitive elements extracted from an image appears to be very important in human recognition. Spatial grouping of primitives into larger units reduces the number of different elements in an image, and allows the elements to be labeled distinctively [Witkin83]. While it can be argued that grouping is important in recognition, it has proven difficult to implement useful grouping operations. Algorithms for performing grouping generally end up being computationally expensive, and prone to error. Recent work by Mahoney [Mahoney87], however, has developed a promising computational

Figure 24. A set of edge fragments not generally recognizable as an object.

framework in which to specify and implement spatial grouping algorithms.

Lowe [Lowe85] presents a nice example illustrating the importance of spatial grouping operations in human visual recognition, and how grouping failure can result in recognition failure. The line segments in Figure 24 do not suggest an object to most people, even after several minutes. By the addition of a single line segment however, as illustrated in Figure 25, the object becomes obvious to most viewers almost immediately.

The addition of a single segment allows the edges at the lower right of the image to be grouped into a circle, apparently causing a wheel to be identified, and then the entire bicycle. By grouping edge fragments into larger units of shape, it seems that humans are able to greatly limit the number of possible interpretations of an image that must be considered in recognition.

As this example illustrates, dependency on grouping can make recognition fail if the grouping operation fails. It is interesting to note that some people are able to recognize the image in Figure 24 after several minutes, which is presumably spent considering a large number of possibilities. Thus if grouping fails it should be possible to spend more effort considering possible hypotheses in order to eventually recognize the object.

A recognition system can make use of grouping operations in several different ways. It is possible to define features that are the result of grouping operations. For example, LFF [Bolles82] and SCERPO [Lowe87] use groups of primitives as features. The bicycle example also illustrates this case, because the circle corresponding to the wheel is a group of primitive segments, and is also

a critical feature for recognizing the bicycle. This use of grouping operations requires that the likelihood of failing to find a group, or of finding a false group, must be relatively low. Otherwise the recognizer will not have reliable features to use for matching.

For a recognition method like alignment, where small sets of local features are used to compute possible matches, a grouping operation can be used to find features that are likely to come from the same object. Sets of features can either be chosen only from the same group, or the groups can be considered before trying arbitrary sets of features. This use of grouping is relatively insensitive to false positive groups, because a group is just a set of features that may correspond to a single object. As long as the groups are smaller than the entire set of image features, the grouping operation will save search in the matching process. Thus it is possible to exploit much less accurate grouping operations in this manner.

Figure 25. Adding a single edge fragment to the image in the previous figure.

A grouping process should be specifiable in terms of local operations because the combinatorics of global operations, such as taking sets of features over arbitrary regions, is as bad as recognition in general. In fact the combinatorics of global operations may even be worse than for recognition, because in current approaches to recognition specific models are used to constrain the search. For instance, a grouping algorithm that considers all triples of features in an image may do more work than an alignment-based recognizer.

Smooth local symmetries [Brady84] can be used to group edge contours that define symmetric regions. The processing performed to find symmetries

is global, however, requiring all pairs of edge pixels to be considered, which is $O(n^2)$ operations for n edge pixels. In cluttered images, numerous "symmetries" can be found that are only due to the juxtaposition of different objects. Thus there will tend to be many false groups. Furthermore, symmetries are sensitive to partial occlusion, and thus are not a very reliable grouping mechanism.

The LFF [Bolles82] and SCERPO [Lowe87] systems both use proximity to group features together, which only involves local computation. In complex images, however, the fact that two features are close together does not mean that they are likely to be from the same object. It is at least as likely that nearby features are just from nearby objects. For edge-based representations in particular, most of the features of an object are relatively far from one another, with the exception of the neighboring features along an edge contour.

Mahoney [Mahoney87] has recently developed a set of routines for performing what he calls visual chunking, which is the division of an image into sets of edges or other primitives that form spatial groups. These groups have the property that they are quite unlikely to occur at random. The processing is done in parallel using simple locally connected processors, and an effort is made to reduce the number of serial steps which must be performed for each grouping operation. The parallel processing framework he develops looks very promising for specifying grouping operations that are useful for recognition.

The GROPER system [Jacobs88] for 2D recognition performs grouping based on proximity, but uses more complex groups of edges that are less likely to occur at random than are single nearby edges. GROPER uses polygonal object models and linear approximations to edge fragments in an image. One of the feature types is a triple of connected edge fragments that define a convex section, as illustrated in Figure 26a. The spatial orientation of these pairs shown in the second part of the figure is relatively unlikely to occur at random, and is indicative of a convex object in the image. Thus it is a good type of grouping to perform on an unknown image.

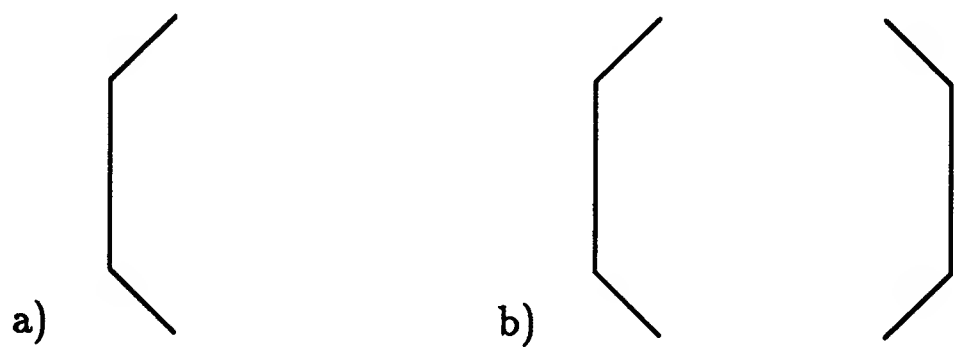

Figure 26. Groups of edge triples can define potential convex regions of an object, a) an edge triple, b) a group.

5.8 Intensity Based Grouping

We have seen that simple, local, edge-based features can be used to compute possible alignments from a model to an image. While edges alone are adequate for performing alignment, to the extent that spatial groups can be extracted from an image, they can also be useful. Spatial regions can be used to define alignment features. For instance, the center of a circle is a spatial attribute that defines a unique alignment point, whereas there is no such point on the contour of a circle. Spatial regions can also be used to find sets of Class II alignment features that are likely to be from the same object. Pairs of these Class II features can then be used for finding possible alignments.

The intensity on each side of an edge segment can be used to form groups of edge segments that are likely to be from the same object. In this section, two different grouping methods are considered. The first method groups together segments where the intensity values on one side of the edge contour are similar, corresponding to the "inside" of a hypothetical object. This method is good for situations where the background changes from one part of an object to another, but the color, texture and illumination of the object are relatively uniform. The second method groups segments where the ratio of the intensity values on both sides of the edge contour are similar. This method is good for the situation where the illumination changes from one part of an object to another, but the color and texture of the object and the background are both relatively uniform.

Both methods average the intensity levels in a small band along either side of an edge segment, in order to provide an estimate of the intensity levels on each side. These values are then quantized and used to cluster segments that have similar intensity levels together, by indexing into a table. To limit the effects of quantization error, each segment is added to those table entries that are within ϵ of the measured intensity value.

In the first grouping method the clustering is performed using the intensity values on each side of a contour. Not all spatial configurations of edges with similar intensity values are equally likely to correspond to the same object. For example, the two edge segments in Figure 27a are much more likely to be part of the same object than are the segments in part b. The shaded side of each segment indicates the side having similar intensity values.

The method checks the relative orientations of pairs of segments with similar intensities. Those pairs where the similar intensities are on facing sides of the segments, as in Figure 27a, are kept. Remaining configurations, such as the one in Figure 27b, are discarded as accidental. Finding pairs of edge segments that are likely to be from the same object is sufficient for the alignment algorithm,

Figure 27. Spatial configurations of edge segments with similar intensities on one side, a) likely to be part of the same object, b) unlikely to be part of the same object.

which only requires pairs of Class II features to solve for a transformation.

The facing sides of two segments are found by connecting the center points of the two edge contours. The side of each segment facing in the direction of the connecting edge is the facing side, as shown in Figure 28a. It is also required that the edge connecting the closer two endpoints of a pair of segments does not cross the edge that connects the farther two endpoints, as illustrated in Figure 28b. If the edges between endpoints cross, as in part c of the figure, then the two segments do not clearly define a region of space.

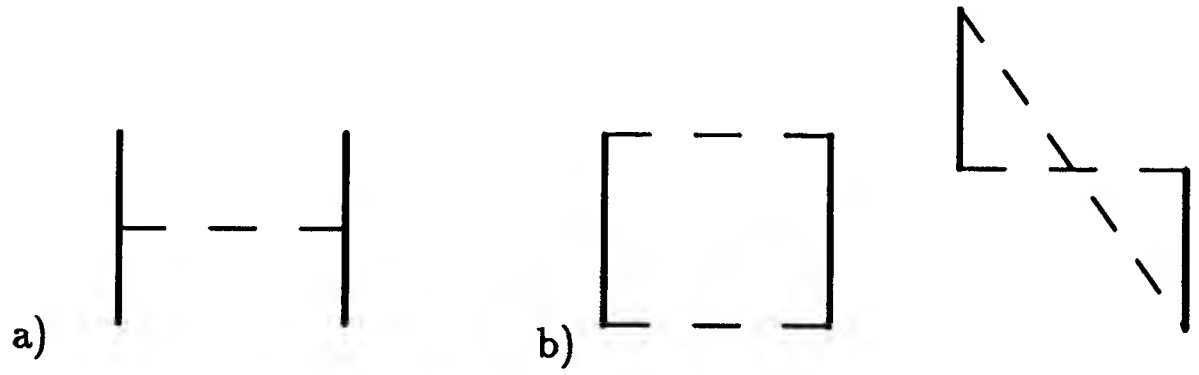

Figure 28. The facing sides of two segments are found, a) using the edge connecting the center points, and b) also requiring the edges connecting the endpoints not to intersect.

The second grouping method uses the ratio of the larger to the smaller intensity value. In this case, two segments match if either the lighter sides of both segments are facing one another, or the darker sides of both segments are facing one another. Otherwise, the two segments are not consistent with varying illumination of a single object.

5.9 Models

For three-dimensional recognition tasks, objects are generally represented in terms of a single three-dimensional model [Besl85]. At the simplest end of the

spectrum are wire-frame or surface models that specify the positions of vertices and edges. More complex models specify the positions of segments [Bolles86] or volumetric or other primitives such as generalized cylinders [Brooks81a] or quadric surfaces.

A single three-dimensional model is not well suited to the problem of visual recognition, however, because it does not explicitly represent information about viewpoint. Only part of a three-dimensional object is visible from any given view, so an image must be consistent with an actual view of an object at a given position and orientation. Thus before a three-dimensional model can be matched to an image, hidden line and surface elimination algorithms must be used to remove the portions of the object that are not visible from a given viewpoint.

It is also difficult to construct a single three-dimensional model from image data, because multiple views must be integrated together. While there are systems for building models from multiple views [Herman84], the problem is quite difficult, and often requires a large number of images. Therefore, most vision systems require object models to be entered explicitly, rather than forming them automatically from images. The very information about viewpoint that is important in recognition is what is discarded in forming a single three-dimensional model from a set of images.

One alternative to representing an object by a single three-dimensional model is to use multiple views. The major issues with the use of multiple views are how many views are needed to represent an object [Koenderink79] [Ikeuchi87], and how to compute the views [Gigas88]. One natural way to choose views for the alignment method is to have a separate view for each viewpoint from which a different set of features are visible. The feature set constitutes the description of an object used in matching, so every time the feature set changes a new view is needed.

Using a separate view every time the feature set changes obviates the need for hidden line and surface computation. The problem of forming models from images is also substantially simplified, because there is no need to integrate multiple views to form a model.

A multiple-view model of a cube, for example, would consist of just one view, with three surfaces visible. Any other view, with just one or two surfaces visible, can be derived from the three-surface view without having to determine and remove hidden parts of the object. In order to model a cube on which each surface has a distinguishing mark, eight views are required, one for each three-surface view (from each vertex). The ORA system uses this type of multiple view model, composed of the three-dimensional locations of the edges and surfaces

visible from a given viewpoint.

While hidden line elimination is not needed for such models, it must be ensured that a model is not oriented in an improper fashion, for example such that the viewing direction is from behind the model. One way to check for proper viewing orientation is to compare the convex hull of the model in its canonical orientation with the convex hull of the positioned model. The convex hull of a planar point set, S, is the smallest convex set of points containing S. Intuitively, it is the shape that a rubber band would take if stretched around the set of points and then released. The convex hull of a set of n points can be computed in time $O(n \log n)$ [Preparata85].

No point on the convex hull of the canonical view should be inside the convex hull in a transformed view, and conversely no point inside the hull in the canonical view should be outside the hull in a transformed view. In other words, points that are extrema of the object should not be moved inside the object by changing viewpoint, and similarly points that are inside the object should not be moved outside it. For example, Figure 29 shows a canonical view of a wedge, and an illegal position where part of the convex hull of the canonical view is inside the hull of the positioned view. The ORA system performs this consistency check for each alignment, and discards matches that improperly orient a model.

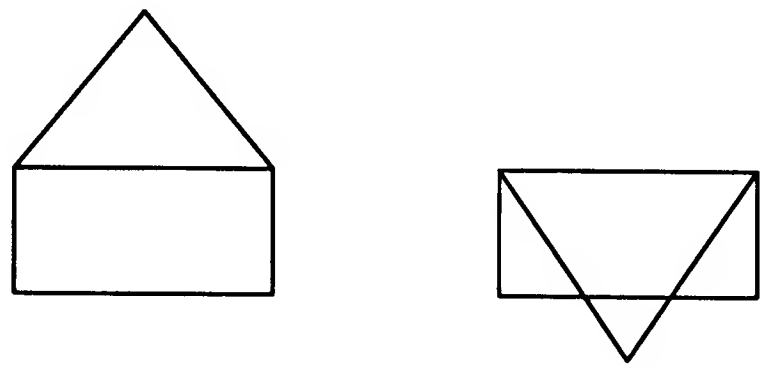

Figure 29. A model of a wedge from a given viewpoint, and an illegal position of that model.

5.10 Chapter Summary

We have seen that features for hypothesizing possible transformations from a model to an image should be

- *reliably computable* given sensor noise and partial occlusion,
- *stable* over viewpoints,
- relatively sparse.

The shape representation developed in this chapter is intended to meet each these goals. The representation is based on edge contours, because they are relatively easy to extract from an image. The features are simple and local, in order to minimize sensitivity to sensor noise and occlusion. Inflection points and straight portions of a contour are used to define edge segments, because these points are preserved under projection. Locally connected sets of the edge segments are then used to define points and orientations for alignment, yielding a representation that is sparse.

Robust methods were developed for computing the curvature and inflections of noisy edge contours. The slope-based least squares method is a fast way to obtain a reliable local orientation estimate for a noisy edge contour. Either the x or y component of the error is minimized as an approximation to the normal error, depending on the slope of the best fitting line. The area zero crossing method is a reliable way to find the zeroes of curvature that correspond to real inflections in noisy curvature data. The significance of a zero crossing is judged based on the area of the curvature function on either side of the crossing.

To the extent that features can be labeled, and groups of features can be found, it may be possible to limit the amount of search required in recognition. Two methods were presented for labeling and grouping features. The first method segments an edge contour at various scales of smoothing to produce a scale-space tree description of the contour. The structure of this tree can then be used to label edge segments at a given scale. This produces more reliable labels than proximity-based labeling schemes, because it only relies on connected pieces of contour. The second method groups edge segments together using intensity information on each side of the edge contour. This method can be used to hypothesize sets of image features that are likely to come from the same object.

The ORA recognition system, described in the next chapter, uses the representation developed here to find features for computing possible transformations from a model to an image. Each feature defines either a point and an orientation, or two points and two orientations. Thus only one or two corresponding model and image features are needed to compute a transformation, using the method presented in Chapter 4.

Chapter 6

The ORA System

This Chapter describes the ORA (Object Recognition by Alignment, pronounced "aura") recognition system. ORA uses singletons or pairs of corresponding model and image features to find possible matches of solid objects to a two-dimensional image. Each transformation from a model to an image is scored by aligning the model with the image, and comparing the transformed model edge contours against nearby image edges. In the worst case the matching process considers $O(m^2i^2)$ transformations for m model features and i image features, because each pair of corresponding features may define a distinct transformation. The time to verify each transformation is approximately $O(m)$, if the length of a model's edge contours is about proportional to the number of features that it contains. Thus the overall worst case matching time is $O(m^3i^2)$.

6.1 System Overview

A grey-level image is first processed using a Canny operator [Canny86] to extract intensity edges. Edge pixels are chained into edge contours by following unambiguous 8-way neighbors. Local neighboring chains are then merged together using a mutual-favorite pairing procedure, described below. Each chain is segmented at inflection points and at the ends of low-curvature regions, as discussed in the previous chapter. The curve segments are then categorized into the classes: `straight`, `corner`, or `arc`. A zero curvature segment is classified as `straight`. A curved segment that has a local high curvature portion, indicative of an orientation discontinuity, is classified as a `corner`. The remaining segments are classified as being an `arc`.

Groups of one or more connected edge segments are used to form alignment features. Each alignment feature defines either a triple of points (a Class I feature) or a point and an orientation (a Class II feature). A single Class I feature in a model and a corresponding feature in an image are sufficient to hypothesize a possible transformation mapping the model into the image. For Class II features, two corresponding model and image features are required.

Each Class I feature in an image is paired with the matching Class I features of a model. Every pair determines a small number of transformations from the model to the image. If a transformation matches more than a certain percentage

of a model's edge contours to image edges, then it is kept as a correct match. Those image features that are accounted for by a correct match are marked so that they need not be considered in subsequent matches. Once all the Class I features have been tried, pairs of unaccounted for Class II image features are matched against corresponding pairs of model features. Possible alignments are computed, and these alignments are verified.

To the extent that any grouping or classification of features can be done, the number of matches considered can be reduced by using only specific pairs of features rather than taking all pairs. As noted in the previous chapter, the grouping process does not need to be very sophisticated in order to be useful. It is enough to find pairs of Class II image features that are likely to be from the same object (although are not necessarily from the same object). These pairs of Class II features can then be used to hypothesize possible alignments, before having to try arbitrary pairs.

Each alignment computation is independent of all the others, so the alignments can all be computed in parallel on a massively parallel machine such as the connection machine [Hillis86]. This computation is constant time as long as the number of alignments to be computed does not exceed the number of processors. Algorithms for a parallel version of the ORA system are discussed later in this chapter.

6.2 Finding Edge Contours

Intensity edges are found using Canny's edge detector [Canny86]. The output of the edge detector is a binary array indicating the presence or absence of an edge at each pixel. In order to be used for recognition, these individual edge pixels must be grouped together into chains that form edge contours. When a given pixel has only one or two neighbors, then the tracing process in unambiguous. Otherwise, a decision must be made as to which pixels belong together as part of an edge.

ORA first forms contours by chaining together pixels in the edge array that have unambiguous 8-way neighbors. Whenever a pixel has more than two neighbors, the current chain is finished, and a new chain is started for each neighbor. The resulting set of chains is then used as the input to an iterative merging procedure. Each step of the iteration finds chains that can be merged together and merges them. When an iteration step performs no merges, the process terminates.

The merging procedure uses what I call *mutual favorite* pairing. Given a set of elements, X, each $x_i \in X$ has a list of potential matches, $m(x_i)$, ordered

from best to worst. For each x_i, call the first element in $m(x_i)$, m_i. For a given x_i, if the first element of $m(m_i)$ is x_i, then x_i and m_i rank each other as the best possible match, and they are paired together.

In the case of edge chains, a given chain, E, can potentially be merged with other chains that have nearby endpoints. If b is one endpoint in an edge chain, E, then any other chain with an endpoint that is within d pixels of b is considered as a possible merge, and similarly for the other endpoint of E. The magnitude of d depends on the length of the chain. In the current implementation $d = \max(5, l/10)$, where l is the length of E.

The candidates for merging with a given chain, E, are ordered based on how well each candidate would merge with E. Consider two edge chains E_1 and E_2 with points e_1 and e_2 taken slightly back from their endpoints, and with the unit tangent vectors to the chains at these two points being $\mathbf{e}_1$ and $\mathbf{e}_2$, respectively, where the tangent vectors point towards the end of the chain, as illustrated in Figure 30. The two points are chosen back slightly from the ends of the chains because the accuracy of edge detectors is generally not very good right at the end of an edge.

Let $\mathbf{v}$ be the unit tangent vector in the direction of the edge e_1e_2 connecting the two endpoints. The merge difference is,

$$D = |e_1 - e_2|(|\angle \mathbf{e}_1\mathbf{v}| + |\angle \mathbf{e}_2\overline{\mathbf{v}}|)),$$

where $\overline{\mathbf{v}}$ is the unit vector in the opposite direction of $\mathbf{v}$. D accounts for the distance between the endpoints, and the difference between the orientation of the segment connecting the two endpoints and the local orientation of the two chains. The more similar the orientation of the three tangents, and the closer the two endpoints, the smaller D will be, and the better the merge is.

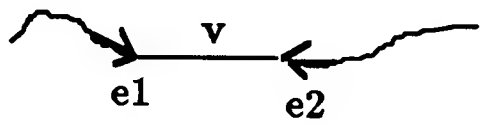

Figure 30. The measure of how well two chains merge takes into account both distance and orientation difference.

Once the candidates for each chain are found and ranked, the iterative mutual favorite pairing procedure starts. One iteration consists of considering both endpoints of each chain. If two chains are paired by the matching procedure then they are merged together, and the merged ends of the chains are removed from consideration in future merges. The iteration terminates when no more endpoints to be merged are found. Clearly the process ends, because there are a finite number of chains, and at each step either two chains are merged into

one, or there are no merges and the iteration stops. Each iteration takes $O(n)$ time for n edges, and at least one chain is removed at each step, so the worst case running time is $O(n^2)$. This worst case only occurs if all the chains have an endpoint near the same place in the image. In general, there are a very small number of chains with endpoints near any given point, so the expected running time is linear in the number of chains rather than quadratic.

6.3 Segmenting a Curve

Edge chains are segmented by finding inflection points and the ends of zero curvature regions, as discussed in Chapter 5. Curvature is computed from the angle between successive tangent vectors to the edge contour. The angle between tangents is corrected for arclength in the case of 8-way connected pixels, where the distance is 1.4 rather than 1.0. The tangent vectors are computed using the slope-based least squares method also described in the previous chapter.

ORA uses the area method, described in the last chapter, to find zeroes of curvature. A zero crossing is kept if the area of the peak on either side of the zero crossing is above some threshold value. Unlike other methods of filtering zero crossings, such as peak height and slope, the area method is relatively insensitive to noise and yet preserves slow zero crossings. This is particularly important for curvature data, where significant slow zero crossings are relatively common.

Given a edge contour, $g(s)$, parameterized in terms of arclength s, its tangent vectors, $\mathbf{T}_s$, and curvature, κ_s, are computed as described in the previous chapter. The significant zero crossings of curvature are then found using the area method. The area between two zero crossings of curvature located at s_1 and s_2 is just the angle between $\mathbf{T}_{s_1}$ and $\mathbf{T}_{s_2}$, the vectors tangent to the edge contour at $g(s_1)$ and $g(s_2)$. Any zero crossing where the magnitude of the area on either side of the zero crossing is above a threshold value is retained, and the others are discarded.

A curve is segmented into positive, negative and zero curvature portions by identifying zero curvature segments, where the curvature is below ϵ, as well as computing inflection points, where the curvature crosses zero. The zero curvature segments are those where the area of the curvature function is below ϵ. The curvature and resulting segmentation of a simple edge contour are illustrated in Figure 31. The zero crossings of curvature and the corresponding points on the contour are marked by dots. As discussed in Chapter 5, this segmentation provides a description of a curve that is relatively invariant under projection, because straight lines always project to straight lines, and inflection points always project to inflection points (or disappear in the case that the contour projects

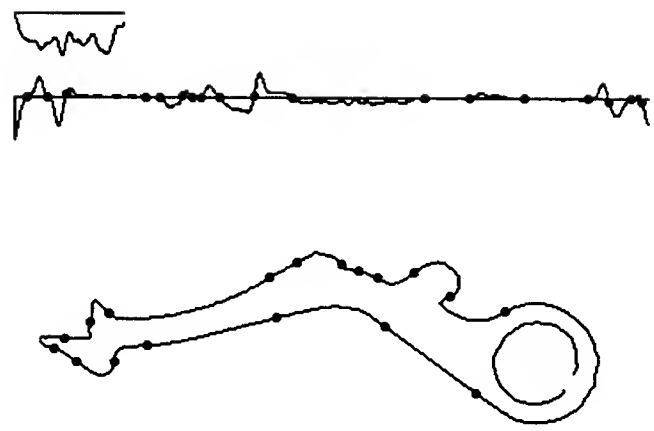

Figure 31. Zero crossings of curvature and the corresponding positive, negative and zero curvature segments of a contour.

to a straight line).

The locations of discontinuities in orientation are also useful for describing a curve, because discontinuities are preserved under projection (except where a curve with a discontinuity projects to a straight line). In a digitized image, however, it is difficult to detect discontinuities. An orientation discontinuity generally involves a large change in orientation, so local high curvature points indicate possible locations of discontinuities. Thus, while high curvature points are not invariant under projection, and thus are bad segmentation points, they are still useful for describing edge segments.

Given a segmentation of a contour into positive, negative and zero curvature segments, it is not sensible to look for high curvature points in zero curvature segments. In the other segments, local high curvature points can be found by looking for peaks in the smoothed curvature, where a peak is defined to be a point s along an arc where the curvature is of greater magnitude than at $s-1$ and $s+1$. The problem with this definition is that a peak is only defined locally. Therefore, a minimum peak height is generally used to remove insignificant peaks. For a segmented contour, there is a natural threshold on the minimum acceptable peak height, namely the average magnitude of the curvature in that segment. The curvature peaks and corresponding edge points are shown for a simple contour in Figure 32. The curvature peaks are marked by dots.

6.4 Features for Alignment

Once a contour has been segmented into positive, negative and zero curvature portions, and the local high curvature points have been identified, the segments are classified into the three types: `straight`, `corner` and `arc`. The zero curva-

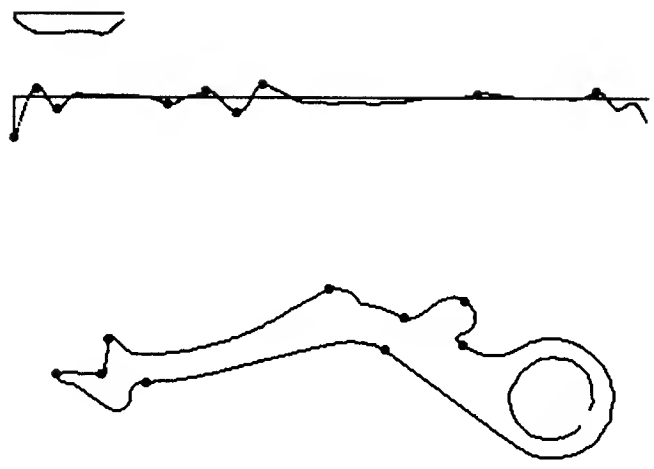

Figure 32. Local peaks of curvature and corresponding points on the edge contour.

ture segments are straight, the segments with high curvature points are corners, and the remaining segments are arcs.

The features used for alignment are sets of points extracted from one or more connected edge segments. Each feature defines either three points (Class I feature) or a point and an orientation (Class II feature). The Class I alignment features, those defining three points, are shown in Figure 33. The features are: i) an edge with two partial edges, ii) a corner with two complete edges, and iii) an arc with two partial arcs (or edges). For those Class I features that define two points and two orientations, a third point is defined where the lines through the points in the specified orientations intersect, as described in Chapter 5. These induced points are indicated by dashed lines in Figure 33.

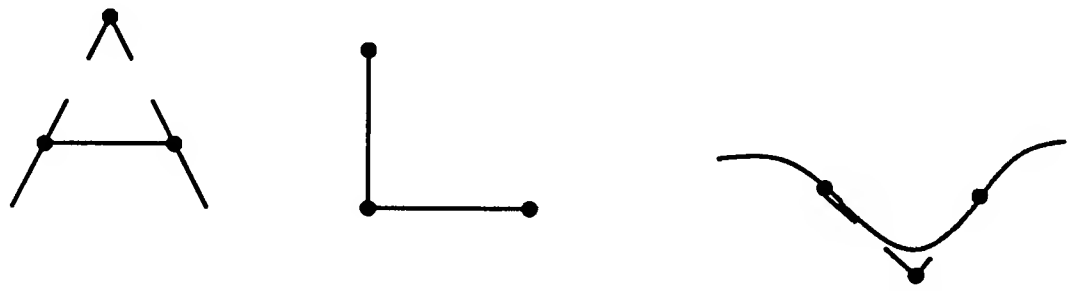

Figure 33. Class I features define three points. A single matching feature is sufficient for alignment.

The Class II features, those defining a point and an orientation, are shown in Figure 34. The features are: i) a corner with two partial edges, and ii) an inflection joining two arcs.

Alignment features are found by considering n-tuples of successive segments along an edge contour as potential features. For example, a Class I feature

Figure 34. Class II features define a point and an orientation. A pair of matching features is sufficient for alignment.

based on three corners can be defined by the five successive segments shown in Figure 35. The end of each segment is marked by a dash, and the three points for the alignment feature are marked with dots. The number of possible such alignment features is linear in the number of edge segments.

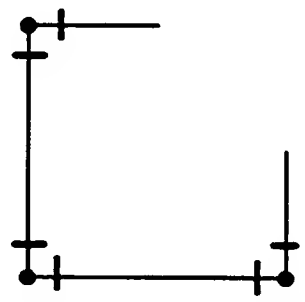

Figure 35. Deriving a three corner Class I feature from successive edge segments.

Image features are additionally assigned a salience measure which can be used to order the features for matching on a serial machine. The salience of an image feature increases with the arclength of the edge contour from which the feature points were extracted, and the closeness to the center of the field of view.

Model features are computed separately for each view of an object. Rather than segmenting the three-dimensional contours, a view is orthographically projected into the $x - y$ plane (the viewing direction for models is the z-axis) and the resulting contours are segmented. The correspondence of image contours to model contours is retained, and the image segments are then used to specify three-dimensional model segments.

6.4.1 Labeling Segments

As discussed in the previous chapter, a hierarchy of curve segmentations can be obtained by smoothing the curvature at different scales, and using the smoothed curvature to segment the edge contour. The segment boundaries are at zero crossings, so each coarser scale segment corresponds to one or more neighboring segments at the next finer scale, forming a scale-space tree [Witkin85].

This scale-space edge contour description is used to label edge segments according to their structure at coarser scales of smoothing. The purpose of the labeling is to allow the matching algorithm to order or prune the space of possible corresponding model and image features. Any pruning on the basis of feature labels may result in missing a correct solution, because of errors in the labeling process. Therefore the current implementation of ORA uses the labels only to decide which pairs of model and image features to consider first.

Some existing recognition systems form distinctive labels using neighboring features to describe a given feature. The problem with this, however, is that an image feature may be labeled using features that are not part of the object being recognized (e.g., as in LFF [Bolles82] and SCERPO [Lowe87]). The scale-space tree provides a more limited form of context, which is less sensitive to errors due to juxtaposition of objects.

6.5 Trying Possible Alignments

Once the alignment features have been extracted from an image, all pairs of Class I model and image features are used to solve for potential transformations. Each transformation is verified by comparing the transformed edge contours of the model with the image. If a transformation maps a sufficient percentage of a model's edges onto image edges, then it is accepted as a correct match. Each alignment feature in the image that is accounted for by a correct match is removed from further consideration by the matcher.

The exact algorithm is given below, where the variable `TRANSES` is used to accumulate the accepted transformations, `IMAT` is a boolean table indicating the image features that have already been matched to some model feature, `MCLASS1` returns the Class I features of a model, and `ICLASS1` returns Class I features in an image. The procedure `CONSISTENT` checks that a model and image feature have consistent labels. `ALIGN1` computes the possible alignment transformations from a pair of Class I model and image features, using the method developed in Chapter 4. `VERIFY` returns all the matching model and image edges given a transformation, `GOOD-ENOUGH` checks that a match accounts for sufficiently many model edges, and `MATCHED-FEATURES` returns the image features accounted for by a match.

```
for MFEAT in MCLASS1(MODEL)
  do for IFEAT in ICLASS1(IMAGE)
       do if CONSISTENT(MFEAT,IFEAT) and not IMAT(IFEAT)
            then for TRANS in ALIGN1(MFEAT,IFEAT)
                   do MATCH ← VERIFY(TRANS,MODEL,IMAGE)
                      if GOOD-ENOUGH(MATCH)
                        then put TRANS on TRANSES
                             for IFEAT in MATCHED-FEATURES(MATCH)
                               do IMAT(IFEAT) ← true
                               od
                   od
       od
  od
```

Once the Class I model and image features have been exhausted, all pairs of Class II model and image features that have not already been accounted for (are not marked in IMAT) are used to solve for potential transformations. The procedure ALIGN2 takes two pairs of model and image features and solves for the alignment transformations. The functions MCLASS2 and ICLASS2 return the pairs of Class II model and image features, respectively.

```
for MFEAT1,MFEAT2 in MCLASS2(MODEL)
  do for IFEAT1,IFEAT2 in ICLASS2(IMAGE)
       do if CONSISTENT(MFEAT1,IFEAT1) and CONSISTENT(MFEAT2,IFEAT2)
             and not IMAT(IFEAT1) and not IMAT(IFEAT2)
            then for TRANS in ALIGN2(MFEAT1,MFEAT2,IFEAT1,IFEAT2)
                   do MATCH ← VERIFY(TRANS,MODEL,IMAGE)
                      if GOOD-ENOUGH(MATCH)
                        then put TRANS on TRANSES
                             for IFEAT in MATCHED-FEATURES(MATCH)
                               do IMAT(IFEAT) ← true
                               od
                   od
       od
  od
```

The alignment computation specifies two possible transformations that differ by a reflection of the model about the three model points. As noted in the previous chapter, a further ambiguity may be introduced by not knowing the exact correspondence between the points in the model feature and the points in the image feature. For instance, a model feature consisting of an edge connected to two partial edges can have two possible correspondences with an image feature.

Sometimes it is possible to discard one of these two possible correspondences based on the kind of partial edges (arc versus straight) at each end. Thus for a pair of Class I model and image features (procedure `ALIGN1`) there are either two or four possible transformations from the model to the image.

To verify an alignment (procedure `VERIFY`), each model edge segment is transformed into image coordinates in order to determine if there are corresponding image edges. The image edges are stored in a table according to position and orientation, so only a small number of image edges are considered for each model edge. The details of the verification process are described in the next section.

When an alignment matches a model to an image, each image feature that is matched by a model feature is taken to be accounted for by that model, and is removed from further consideration by the matcher (by marking it in `IMAT`). This can greatly reduce the set of matches considered, at the cost of missing a match if image features are incorrectly incorporated into verified alignments of other objects. This is unlikely, however, because the verifier has a low chance of accepting false matches.

The worst case running time of the matching process is $O(m^3 i^2)$ for m model features and i image features. This is because all the features may be of Class II, and it may be necessary to consider matching each pair of model features against each pair of image features, which is $m^2 i^2$. Scoring each alignment then involves transforming the model edges to image coordinates, which is about $O(m)$ operations. In practice, however, it is not necessary to consider all pairs of features, because each correct alignment will eliminate a number of image features from consideration.

6.6 Verification

Once a potential alignment transformation has been computed, it must be determined whether or not the transformation actually brings the object model into correspondence with an instance in the image. This is done by transforming the model to image coordinates, and checking that the transformed model edges correspond to edges in the image. The verification process is very important because it must filter out incorrect alignments without missing correct ones. In Chapter 3 we saw that simple verification procedures, such as counting the number of transformed model points that lie near some image point, are inadequate for complex images.

Before verifying a transformation, the system checks that the transformation specifies a valid orientation of the model. As discussed in Chapter 5, this

involves comparing the convex hull of the model in its canonical view with the convex hull of the model in the transformed view. Points inside the convex hull in one view must not be outside the convex hull in the other view.

The verification method is hierarchical, starting with a relatively simple and rapid check to eliminate many false matches, and then continuing with a more accurate and slower check. The initial verification procedure compares the segment endpoints of the model with the segment endpoints of the image. Recall that a segment endpoint is defined at inflections and the ends of straight segments. Each point thus has an associated orientation, up to a reflective ambiguity, defined by the orientation of the edge contour at that point. A model point correctly matches an image point if both the position and orientation of the transformed model point are within allowable error ranges of a corresponding image feature, as illustrated in Figure 36. Those alignments where a certain percentage (currently half) of the model points are correctly matched are then verified in more detail.

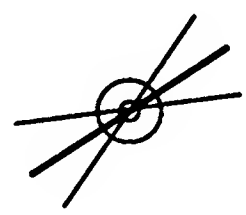

Figure 36. The initial verification of a match compares model segment endpoints to image segment endpoints, requiring both the location and orientation difference to be small.

The detailed matching procedure compares the entire edge contours of a transformed model with the image. Each segment of the transformed model contour is matched against the image segments that are nearby. The comparison of contours is relatively time consuming, but makes it very unlikely that an incorrect match will accidentally be accepted. The initial verification generally eliminates many hypotheses, however, so this more time consuming procedure is only applied to a small number of hypotheses.

The comparison of a transformed model segment with an image is intended to distinguish between an accidental and a non-accidental match. For instance, an accidental match is very unlikely if a transformed model segment exactly overlaps an image segment such that they coterminate. If the image segments are nearby, or do not coterminate with the model segment, then there is a higher likelihood that the match is accidental. There are three kinds of evidence for a match: positive evidence, neutral evidence, and negative evidence, as illustrated in Figure 37. A model edge that lies near and coterminates with one or image edges is positive evidence of a match, as shown in the first part of the figure.

A model edge that lies near a very long or very short image edge is neutral evidence of a match, similar to there being no nearby edge. A model edge that is crossed by one or more image edges that, at a sufficiently different orientation than the model edge, is negative evidence of a match.

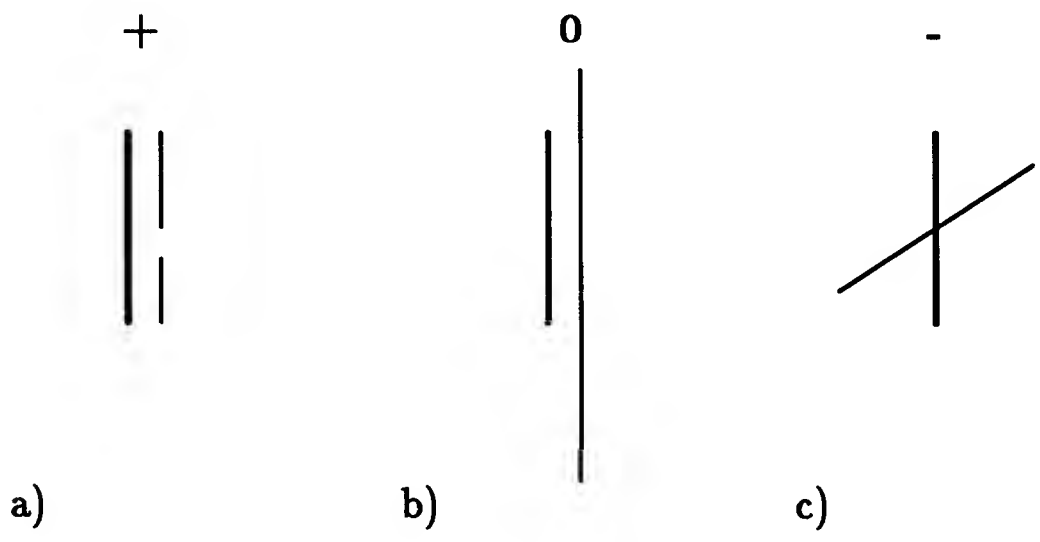

Figure 37. Matching model segments to image segments, a) positive evidence, b) neutral evidence, similar to no matching edge, c) negative evidence.

The verification process uses a table of image edge segments stored according to x position, y position, and orientation. A given image segment, i, is entered into the table by taking each point, $g(s)$, along its edge contour and the corresponding tangent orientation, $\mathbf{T}_s$, and storing the triple $(i, g(s), \mathbf{T}_s)$ in the table using quantized values of $g(s)$ and $\mathbf{T}_s$. A positional error of 10 pixels and an orientation error of $\frac{\pi}{10}$ radians are allowed, so a given point and orientation will generally specify a set of table entries in which to store a given triple.

Only those image segments of compatible types are considered as possible matches to a transformed model segment. A straight edge in the image is compatible with all three types of model edge segments (straight edge, arc and corner) because it can be the projection of any of these segments. A corner in the image matches only against a corner in the model, and similarly for an arc in the image.

In order to match a model segment to an image, each location and orientation along the transformed model contour is considered in turn. A given location and orientation are quantized and used to retrieve possible matching image segments from the table. These segments are then filtered to remove the ones with incompatible types. Given a location, s, along a transformed model contour, with the position, $g(s)$, and the corresponding tangent orientation, $\mathbf{T}_s$, each image triple retrieved from the table, $(i, p_i, \mathbf{T}_i)$, is ranked using the function,

$$d = |p_i - g(s)||\angle \mathbf{T}_s \mathbf{T}_i|.$$

If the best matching image triple is at a distance of less than 10 pixels and has an orientation difference of less than $\frac{\pi}{10}$ radians, then the location s is accounted for by that image triple. Rather than considering each location, s, along the model contour, it is possible to consider every n-th location, reducing the accuracy but speeding up the match.

The result of this processing is a list that associates each location, s, along a transformed model segment with the best matching location along some image segment (if any). The next step is to take each image segment in this list and determine what percentage of that segment's contour is accounted for by entries in the list. If a significant portion of an image contour is accounted for (currently 50 percent) then the image contour matches the model segment, otherwise the image contour is taken to have matched accidentally, and is removed from the match. After removing the accidentally matching image segments, the percentage of the model contour that is matched by the image is determined. This percentage is the positive evidence that the model segment is actually in the image.

This matching procedure requires a good correspondence between a transformed model segment and one or more image segments, rather than just considering how much of a model contour lies near some image contour. The procedure also allows multiple image segments to correspond to a given model segment, which is important in real images where edges often become broken due to shadows, sensor noise, or occlusion.

Negative evidence is used to further limit the chance of verifying incorrect alignments. If a transformed model segment crosses some image segment then it is unlikely that the hypothesized position of that model segment is correct. Crossing segments are found similarly to potential matching segments, except that the table for looking up crossing segments indexes only on position, not orientation. If an image segment is found that crosses a model segment, other than within ϵ pixels of the ends of the contours, then the model segment is taken not to match the image, regardless of the positive evidence for that segment.

In summary, the verification method uses local information (the locations and orientations along an edge contour) in a transformed model segment to look up potentially matching image segments. Those matching image segments that are well accounted for by the model segment (match half their arclength to the model segment) are retained, and the others are discarded as accidental matches. The remaining matching image segments are then used to determine the percentage of the model contour that is accounted for by the image.

The verification procedure is designed to be able to find any evidence in support of a hypothesized alignment, without accepting alignments for which

there is only poor supporting evidence. A single model segment can be accounted for by several image segments, allowing for noise and occlusion that may cause a given segment to be split into pieces. By requiring a substantial portion of each image segment to overlap with a model segment, however, it is unlikely that an accidental correspondence will cause an incorrect verification.

6.7 Parallel Algorithms

The alignment computation is performed using one or two pairs of corresponding model and image features. Each of these computations is independent of the others, so all the possible alignments of a model with an image can be computed in parallel. This section outlines algorithms for performing the feature extraction and alignment operations of the ORA system on the connection machine.

The connection machine [Hillis86] is a fine-grained parallel computer with between 16K and 64K processors. The machine is a single instruction multiple data (SIMD) architecture, because one instruction stream is broadcast to all the processors. Each processor is a 1-bit serial ALU. There are two versions of the machine, the CM-1 has 4K bits of memory per processor, and the CM-2 has faster processors, 64K bits of memory per processor, and optional floating point hardware.

The connection machine routing network allows each processor to send a message to any other processor, using the processor's unique integer address. The processors can be viewed as the vertices of a 16-dimensional hypercube, so each processor can communicate directly with 16 other processors, and must route messages to the remaining processors.

The hypercube connections are directly exploited by a set of primitive operations called scan operations. These operations can distribute values among processors or aggregate values using associative operators in about the same amount of time as a message routing cycle. For example, the enumerate operation can assign a unique number to each processor in a set of processors by executing a plus-scan.

An important non-primitive operation is distance doubling, which can be used to compute any binary associative operation on a set of processors linked in a ring or list. For example, doubling can be used to find the extremal value in a list of elements that are stored one element per processor. This operation takes $O(\log m)$ steps for a list of length m. Initially each processor has the address of its neighbor in the list, and at each step, i, a processor gets the address of a processor 2^i+1 away, and the extremum of all the values within 2^i-1 processors. This operation can be used to assign a unique label to each linked list in the

machine, by maximizing the processor address in each list. At the end each processor will have the label of the list to which it belongs.

Vision algorithms on the connection machine make use of three classes of operations: uniform spatial operations, non-uniform spatial operations, and operations on features. Spatial operations view the processors in the machine as forming a two-dimensional grid of pixels. Uniform operations such as smoothing are applied to all groups of neighboring processors, and non-uniform operations are applied only to certain processors. Feature-based operations, on the other hand, assign one or more features to each processor, and perform operations on sets of features.

The Canny edge detector [Canny86] has been implemented on the connection machine [Little87]. On a 16K processor machine, computing the intensity edges for a 256 pixel squared image takes about 150 msec. The Gaussian smoothing, directional derivative computation, and non-maximum suppression are all uniform spatial operations, and are done in constant time. The thresholding operation is a non-uniform spatial operation, and requires time $O(\log m)$ in the longest edge chain, using the distance doubling operation to propagate values along a chain.

Linking together neighboring pixels in the Canny output to form edge contours is spatially uniform, only requiring communication among neighboring processors. Chains of pixels are then formed using distance doubling to assign a unique label to each linked list of neighboring pixels.

Computing orientation and curvature, and smoothing the curvature are all local operations along the edge chains, and thus can be done in constant time. The zero crossings of smoothed curvature can be locally detected by comparing neighboring processors in the edge chains. In order to threshold the zero crossings, the area of the curvature function between zero crossings must be computed. This can be done by propagating the tangent orientation to the contour at each zero crossing to the next zero crossing in the chain, and then computing the angle between pairs of tangents. Using doubling this takes time $O(\log m)$ in the longest distance between two zero crossings. Zero crossings with low corresponding areas are removed locally. The zero crossings can then be used to form segments by propagating the highest processor address in the chain between two zero crossings as a unique label for that segment, using doubling.

Once the segments have been formed, the processing switches from spatial to feature based operations, because the segments are the primitive features of the matching process. To collect the points in each feature into a single processor requires time linear in the longest feature chain, because the edge contour points of a given feature must be collected into an ordered list in a single processor.

Given a set of image features, finding all the possible alignments of a model requires taking the product of the set of model features and the set of image features. If each of the pairs is assigned a single processor, then all the alignments can be computed in parallel. The product of two sets can be computed in constant time [Blelloch88]. The alignment operation then involves mathematical operations local to each processor, and thus can also be done in constant time.

Thus the extraction of alignment features and the computation of possible alignment transformations are both well suited to implementation on a massively parallel architecture such as the connection machine.

6.8 System Summary

Initially an image is processed to extract edges [Canny86], and the edge pixels are chained into contours. The local curvature of the contours is computed, and zeroes of curvature are used as segmentation points. The resulting edge segments are relatively stable over changes in position and orientation, because zeroes of curvature are preserved under projection. Locally connected groups of these edge segments are used as features for alignment. Each feature defines either a point and an orientation, two points and two orientations, or three points.

Pairs or singletons of corresponding model and image points are used to compute possible transformations from a model to an image, using the method developed in Chapter 4. Only one or two corresponding model and image features are needed to specify a transformation that is unique up to a reflection. Each transformation is then used to align the model with the image, by transforming the model into image coordinates.

A transformation is verified by comparing the aligned model edges with image edges. Positive evidence, such as proximity and cotermination of model and image edges, is used to find edge contours that are accounted for by a match. Negative evidence, such as crossing model and image edges, is used to mark model edge contours as not accounted for. If more than a certain percentage of a model's edge contours are matched, then a transformation is accepted as specifying a correct match of the model to the image. The verification process is hierarchical. First local model and image points are compared, and then the entire edge contours are traced.

6.9 Some Results

The recognizer has been tested using images of both simple polyhedral objects

and curved laminar objects in relatively complex scenes, taken under normal lighting conditions in offices or laboratories. While the representation of solid objects is not constrained to polyhedra, it is currently difficult to enter the three-dimensional coordinates of non-polyhedral objects in order to form models.

Figures 38 - 42 show the performance of the recognizer on several images of solid objects. Part a) of each figure shows the grey-level image, part b) shows the Canny edges, part c) shows the edge segments (straight edges are in bold and corners are marked by dots), and part d) shows all the verified instances of the models in the image. Each model is projected into the image, and the matched image edges are shown in bold.

Three different models, a cube, a wedge, and an object consisting of a rectangular block and a wedge glued together, were matched to each of the images in Figures 38 - 42. These models are shown in Figure 44, with the straight segments in bold and the corners marked by dots. There are five to ten alignment features for each model, and several hundred alignment features in each images, so many thousands of possible transformation were considered for each example. All the hypotheses that survived verification are shown in part iv) of the examples. The matching time (after feature extraction) for these images is between 2 and 5 minutes on a Symbolics 3650.

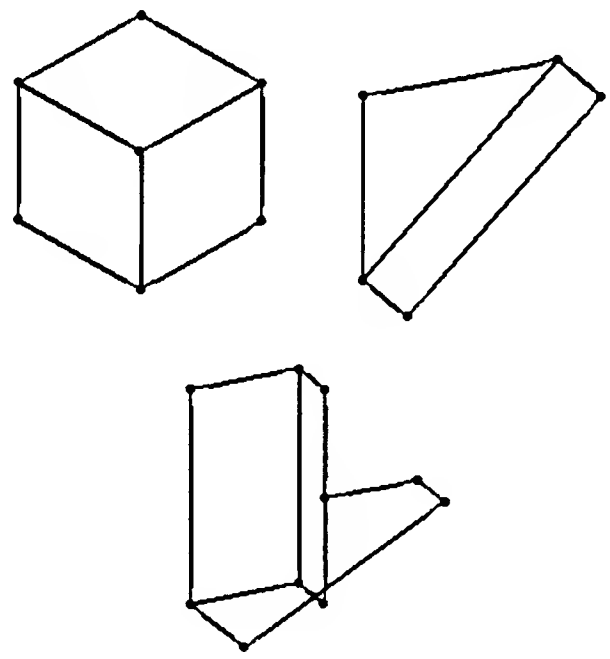

Figure 44. The three solid object models that were matched to the images.

In Figure 39 the wedge model is not very well aligned with the image. The match is good enough to pass the verification stage, but the matching contours are at the outer limit of the error tolerance. The problem can be traced to the fact that the Class I feature used to compute the alignment transformation is the

bottom part of the wedge, which is partly occluded by the block. The occluded portion of the feature is small enough that the match is within tolerance. In order to speed up recognition on a serial machine, the matched image edges are marked as accounted for and are not considered in subsequent matches. If additional features from those image edges had been considered, then a pair of Class II features (such as the two front corners) would have hypothesized a more accurate match.

Because an alignment is computed from only three corresponding points, sensor error in those points causes small errors in the match. Furthermore, the transformation does not take into account perspective distortion. To get a more accurate transformation from a model to an image, the features that are brought into correspondence by the 3-point transformation can be used to compute a least squares solution to the perspective viewing equation (e.g., as in [Lowe85]). Finding corresponding features using the 3-point method minimizes the number of transformations that must be considered, and simplifies the computation of each transformation. Then solving for the least-squares perspective transformation gives a very accurate match.

Figure 43 shows an example of recognizing a laminar object that has curved contours. The model of this object was simply formed from the edge contours of an isolated instance of the object on a high contrast background. This model is shown in Figure 45, with the straight segments in bold, and the corners marked by dots.

Figure 45. The model of the laminar widget.

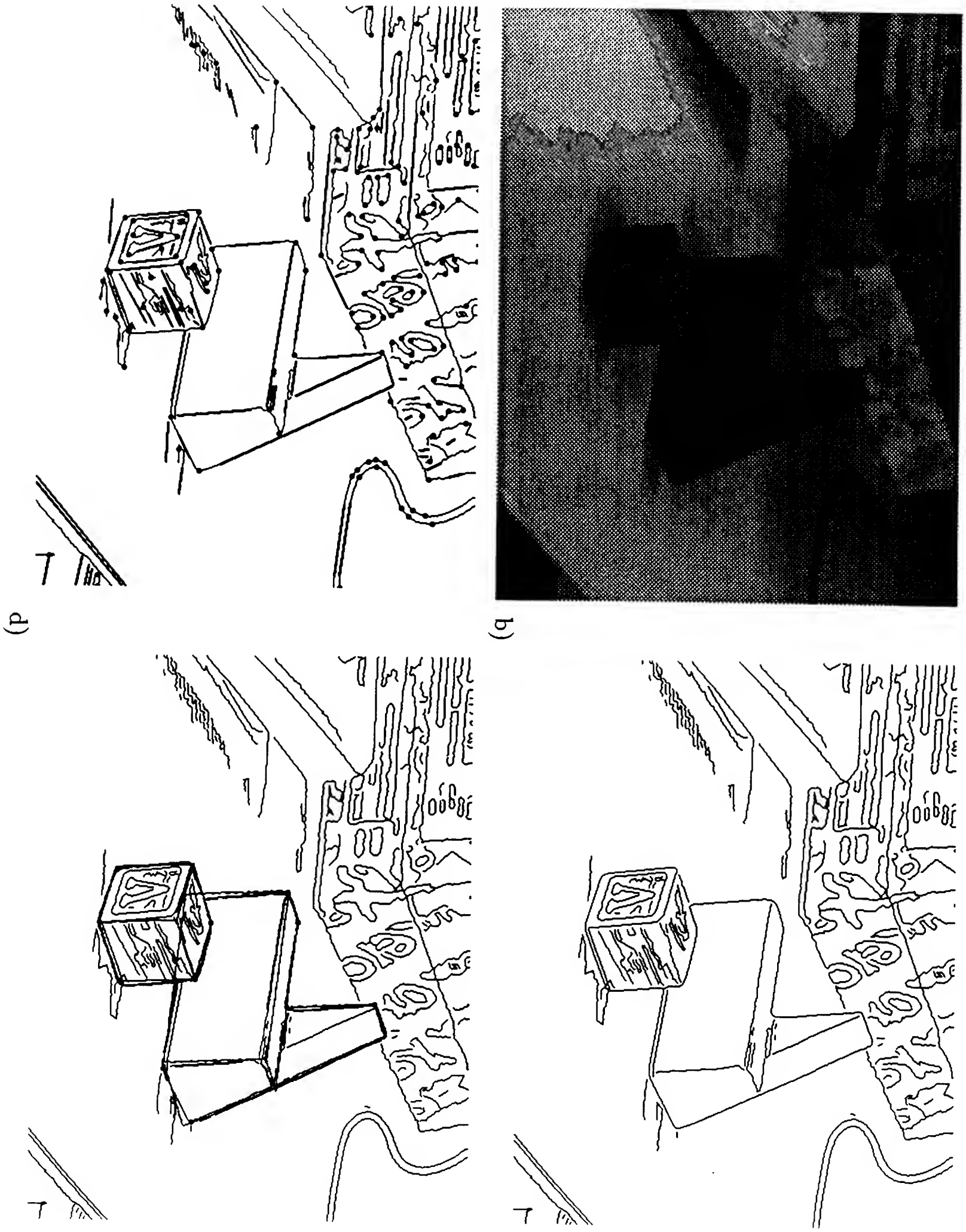

Figure 38. Recognizing solid objects, see the text for an explanation.

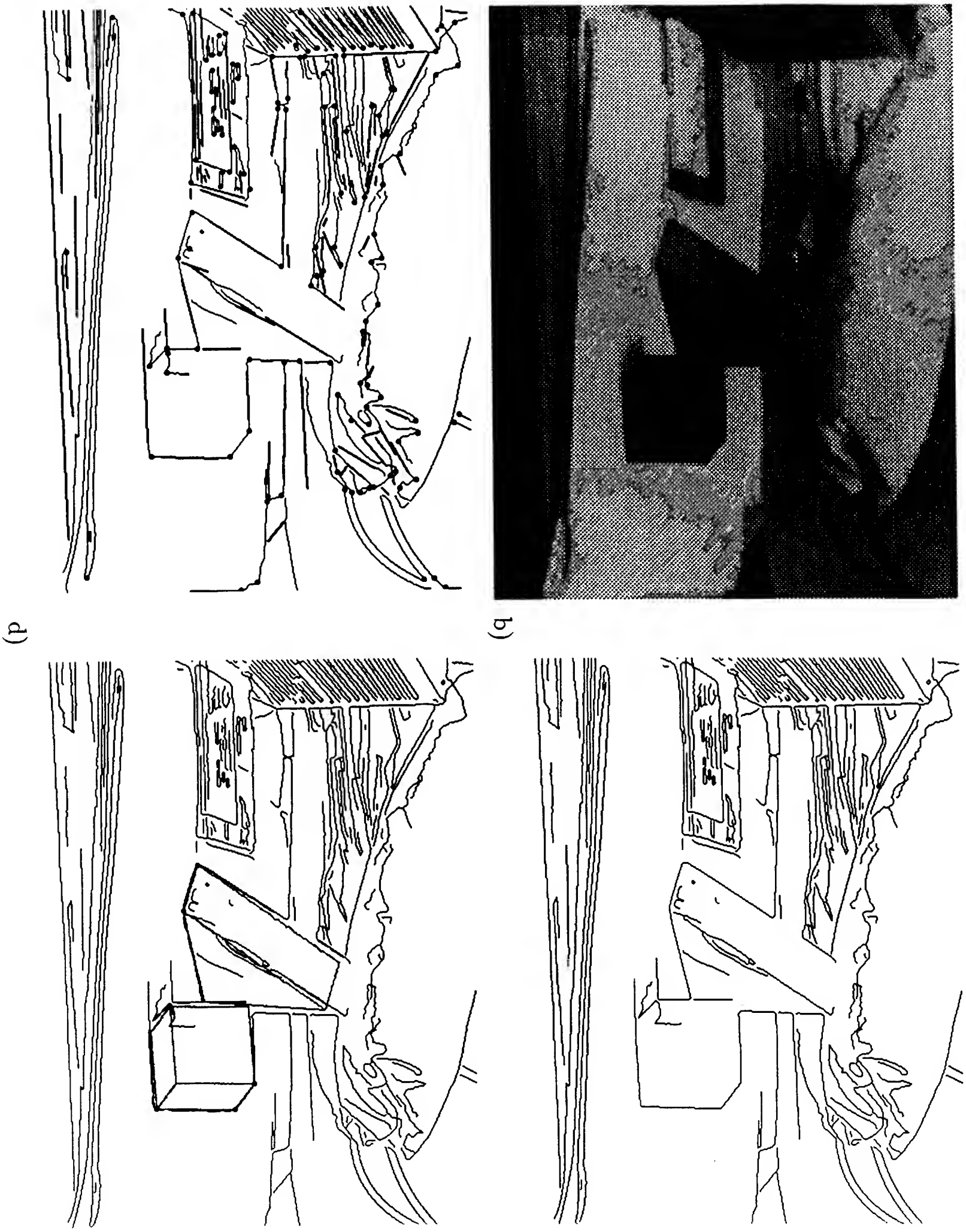

Figure 39. Recognizing solid objects, see the text for an explanation.

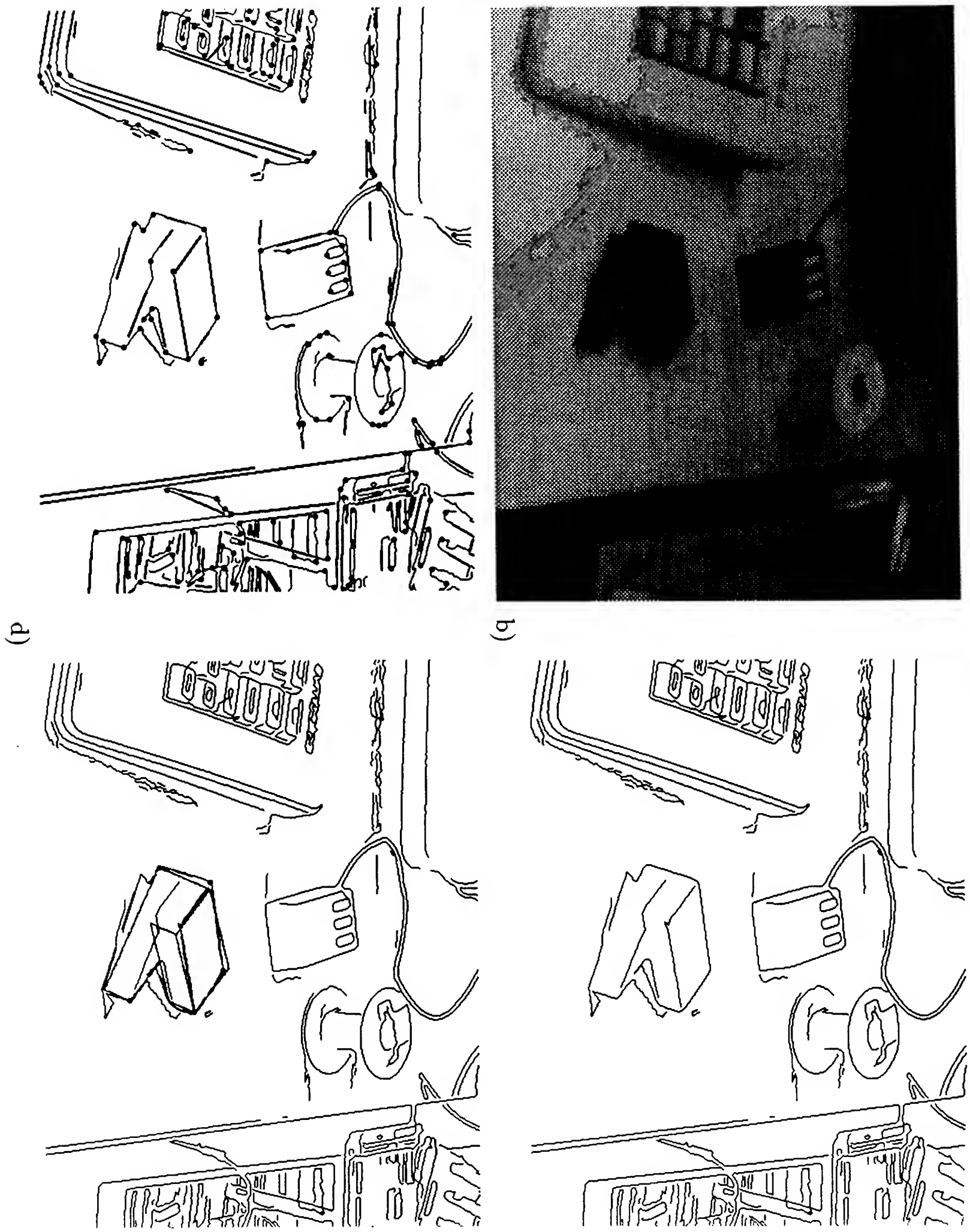

Figure 40. Recognizing solid objects, see the text for an explanation.

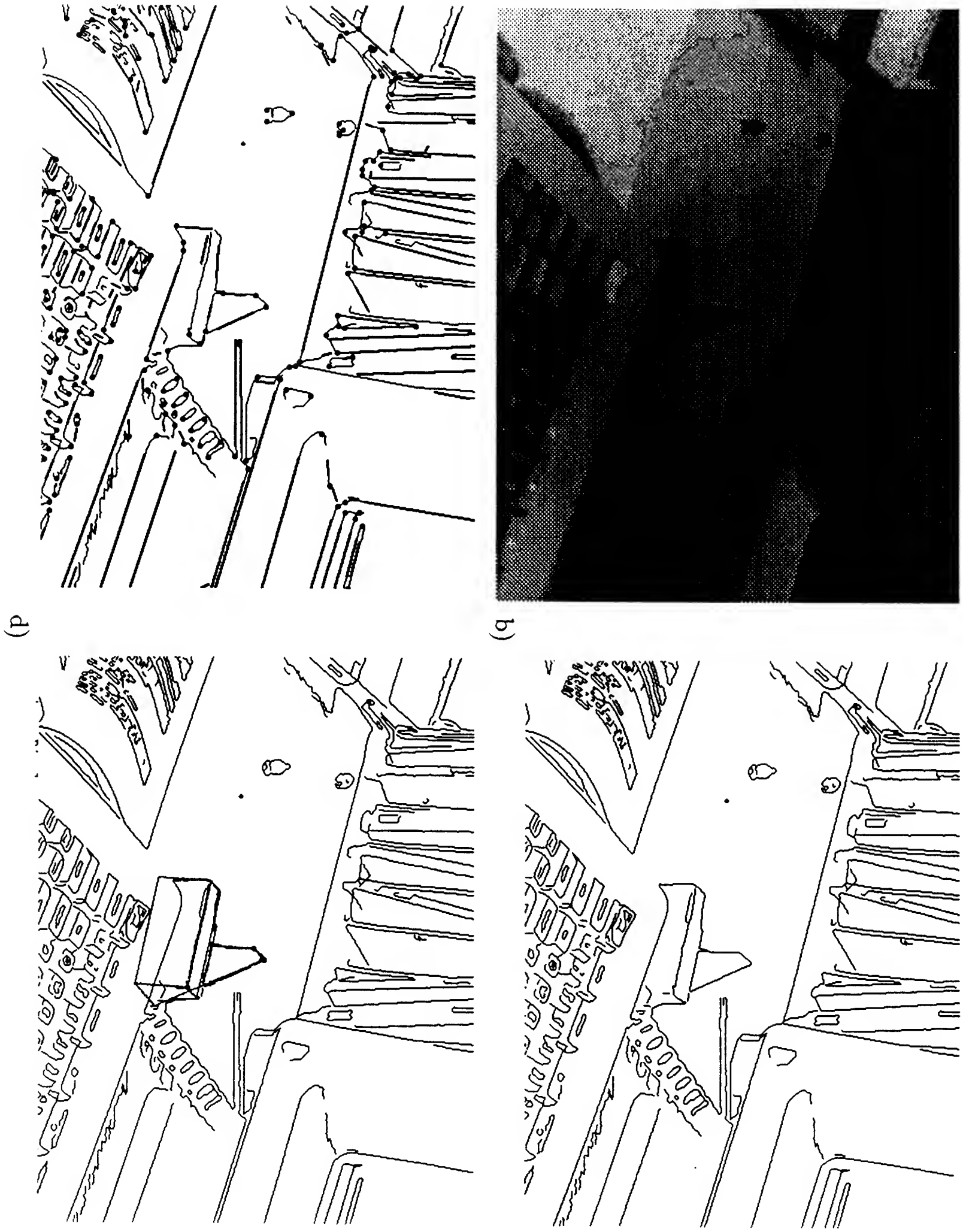

Figure 41. Recognizing solid objects, see the text for an explanation.

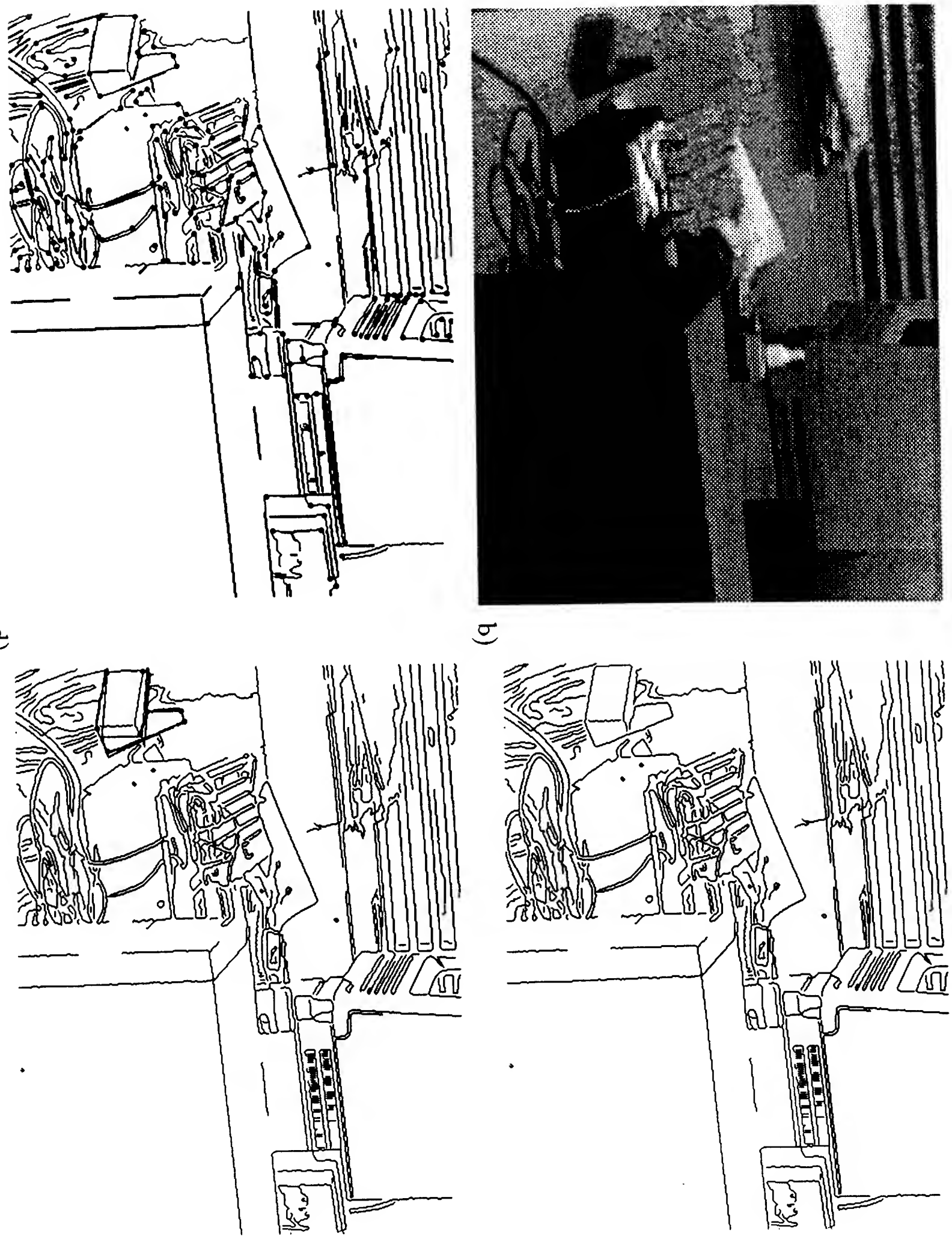

Figure 42. Recognizing solid objects, see the text for an explanation.

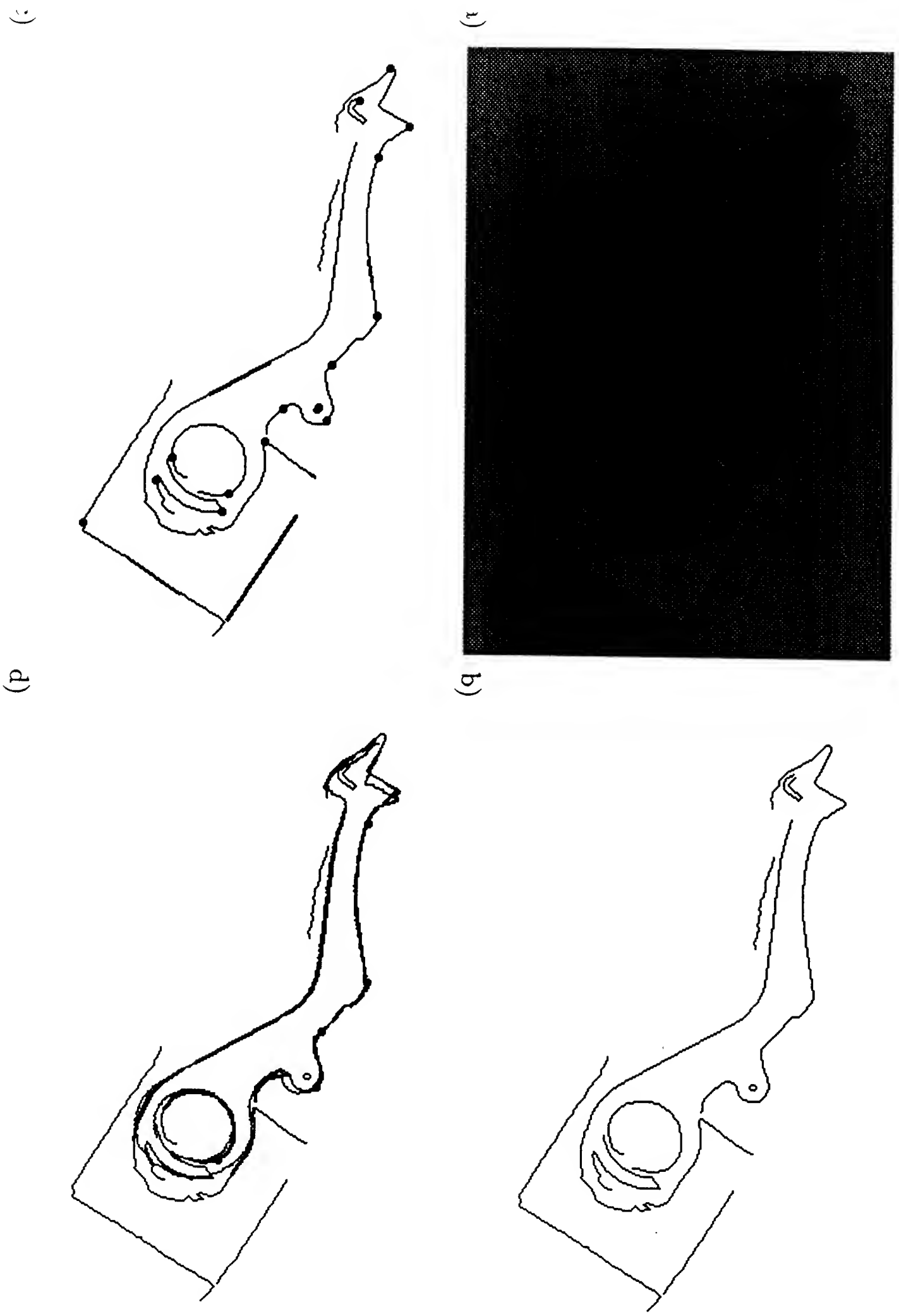

Figure 43. Recognizing a laminar object, see the text for an explanation.

Chapter 7

Aligning Non-Rigid Objects

The alignment method, as presented thus far, assumes that an object is rigid, so that a single similarity transformation will map each model feature onto its corresponding image feature. In the case of a non-rigid deformation, such as an object with moving parts, or an object that has been stretched or bent, the rigidity assumption will be violated. This chapter considers how to extend the alignment method to the problem of recognizing non-rigid deformations of objects, and shows some examples of a particular implementation of the non-rigid alignment method.

In order for shape information to be useful in recognition, the transformation from a model to an image must preserve some attributes of an object's shape. The idea underlying the non-rigid alignment method is that a deformation must be approximately rigid locally in order for it to preserve any shape information. For example, Figure 46 shows an object and an instance of the object that has been bent near the bottom. The parts of the object on either side of the bend are each approximately rigid. As a more extreme example, a nonuniformly stretched object such as a cartoon character on silly putty can be viewed as consisting of local regions over which the model has been scaled. If these regions become infinitesimal, then the deformation no longer preserves shape information.

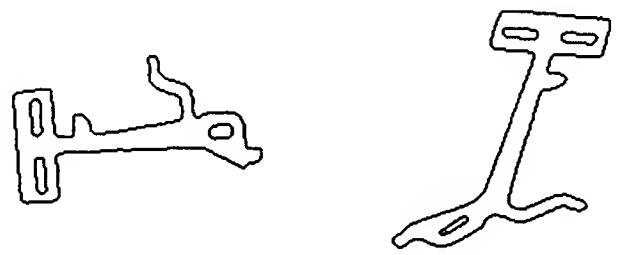

Figure 46. A transformation that is non-rigid but preserves shape information can be viewed as locally rigid. For example, a bent object consists of rigid subparts.

Most recognition systems can only recognize rigid transformations of an object. Those systems that allow for non-rigid transformations do so by parameterizing

an object model so that it specifies a set of rigid parts, and the allowable motions of those parts [Brooks81a] [Grimson87b]. While some objects do deform in predictable ways, an unexpected non-rigid deformation should not preclude recognizing an object. Thus the extension of the alignment method discussed in this chapter allows for parameterized models, but is also capable of recovering non-rigid deformations at recognition time.

If a model specifies a set of rigid subparts, the standard alignment technique can be applied to each subpart. It is then necessary to check that these separate alignments are consistent with a single instance of the object, by ensuring that adjacent parts of the model are also adjacent in the image. The details of the method are discussed below, in the section on combining local matches.

Recovering a non-rigid deformation during the matching process requires finding the pieces of a model that are being rigidly transformed. Each locally rigid piece can be matched using the standard alignment method, and together the pieces approximate the non-rigid transformation. Two methods of finding the locally rigid pieces are considered here. The first method works by tessellating an image into triangles, performing a rigid alignment for each triangle, and using these alignments to iteratively refine the tessellation. The second method works by finding rigid partial matches between a model and an image, and then piecing together the partial matches into a non-rigid match.

7.1 Tessellating the Image

An image can be tessellated into regions that correspond to locally rigid parts of an object. By computing a separate alignment for each region, rigidity is preserved locally but not for the object as a whole. Triangulation is a good tessellation pattern for the alignment method, because three points are used to compute a transformation [Ullman87]. An image can be triangulated by using the features in the image to define a point-set, as illustrated in Figure 47. The Delaunay triangulation (the dual of the Voronoi diagram) is a good triangulation method because it maximizes the minimum angle of the triangles. This reduces the number of acute triangles, which are bad for computing alignments because their vertices are nearly colinear. The Delaunay triangulation of a set of n points can be computed in $O(n \log n)$ time [Preparata85].

Given a triangulation of an image, and a potential correspondence between model and image features, each model triangle can be aligned with its corresponding image triangle. If a triangle falls on a rigid part of the object, then the alignment of model and image edge contours inside the triangle will be good. If the match inside a triangle is not good, then either the correspondence of

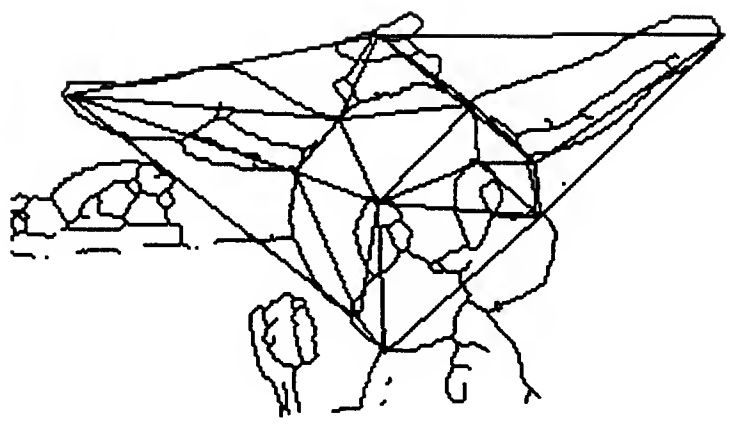

Figure 47. Tessellating an image by triangulating the set of feature points.

model and image features is incorrect, or the deformation inside the triangle is non-rigid. A finer tessellation can be performed to determine if there is a rigid approximation to the transformation.

The size and location of the triangles required to approximate a given non-rigid transformation depends on the particular transformation. A set of rigid triangles can be recovered using an iterative algorithm that starts by computing a standard rigid alignment of a model with an image. If this initial alignment does not match any additional model features to the image, then no further action can be taken. If additional features are brought into correspondence, however, then these features are used to define additional model and image points. The resulting point set is triangulated, and a rigid alignment is performed for each triangle. This process is repeated until either a good match of the model to the image is obtained, or the triangles become too small to reliably estimate the transformation.

There are three measures of how well such a locally rigid alignment matches a model to an image. The first measure is the degree of match between the transformed model and the image. The second measure is the extent to which the model is tessellated. If the triangulation is very fine, then the overall transformation is highly non-rigid, and the match is not as good as a more rigid transformation. Finally, the transformations specified by neighboring triangles should change relatively smoothly from one triangle to the next. If the pattern of transformations is highly discontinuous, then the match is poor.

As an example of this iterative triangulation method for finding a locally rigid alignment, consider matching the two drawings of a bunny rabbit shown in Figure 48. The image on the left is a "model" that will be transformed to match the image on the right.

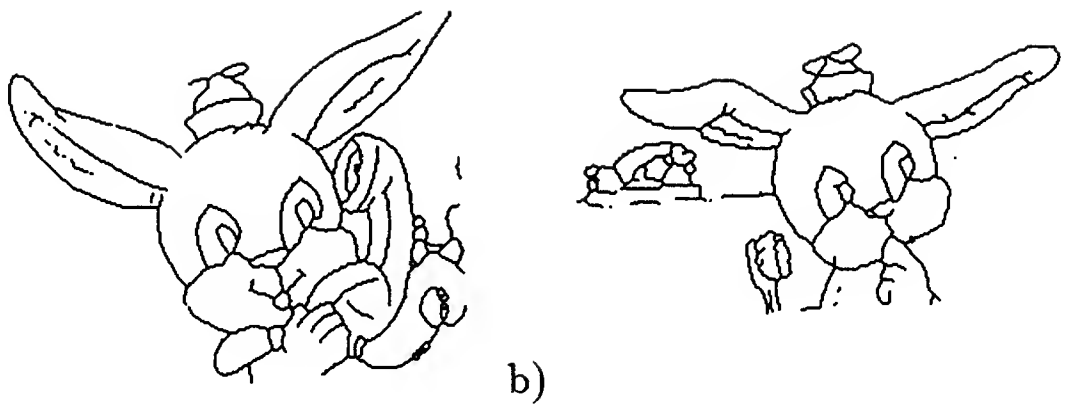

Figure 48. Two cartoons of a bunny rabbit to be nonrigidly aligned.

Figure 49 shows a rigid alignment of the model with the image. The three points, the tip of the hat and the two cheeks, were chosen by hand. Part a) of the figure shows the triangulation of the model points, part b) shows the transformed model, part c) shows the transformed model overlaid with the image, and part d) shows the correlation of the transformed model with the image. The correlation is computed by keeping only those pixels in the transformed model that are within 4 pixels of an image pixel.

The match of the head and the hat is quite good, but the ears do not match well because they are bent in the image and straight in the model. In fact, part of the telephone is preserved in the correlation image because it overlaps with the ear in the image. From the overlay there are several good points to use for iterating the method in order to obtain a better alignment. For instance, the places where the ears join the head are sharp concavities that are at similar positions and orientations in both the image and the transformed model. While in general it is not reasonable to use measures such as degree of curvature and orientation for matching a model to an image, in this case the model and image have already been partially aligned.

Figure 50 shows an alignment using some additional corresponding points that are specified by the initial alignment (the points are again hand picked). From the correlation image it can be seen that this alignment does quite a good job of matching the model to the image. The only part of the rabbit that does not match well is the right cheek, which is occluded by the phone in the "model" image.

In summary, the triangulation method is a coarse-to-fine, iterative technique for finding a set of locally rigid matches between a model and an image that combine to form a non-rigid deformation of the model. The method starts with a standard rigid alignment of the model with the image, and yields a set of local alignments of regions of the model with regions of the image.

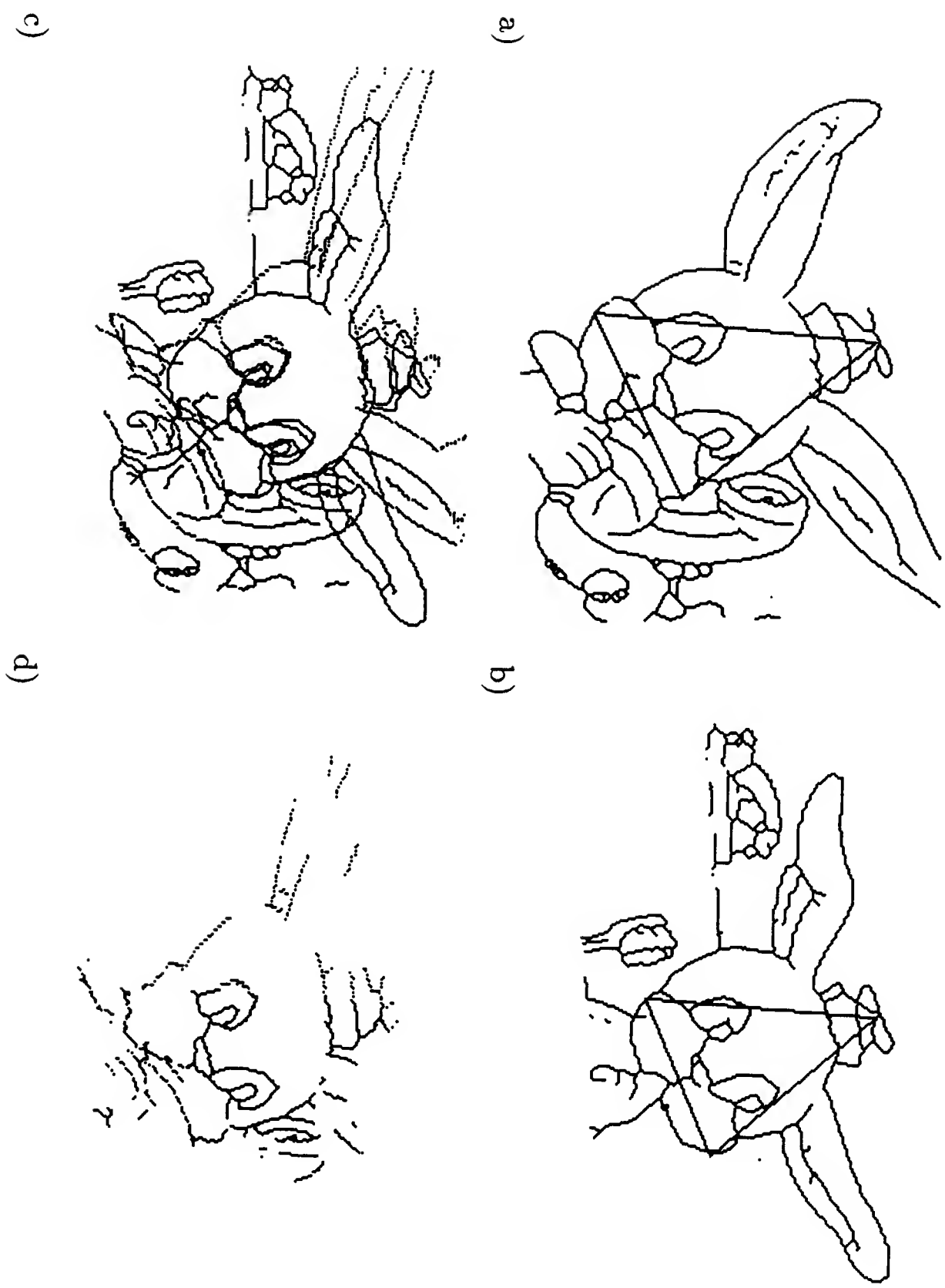

Figure 49. Initial rigid alignment of two images, see the text for an explanation.

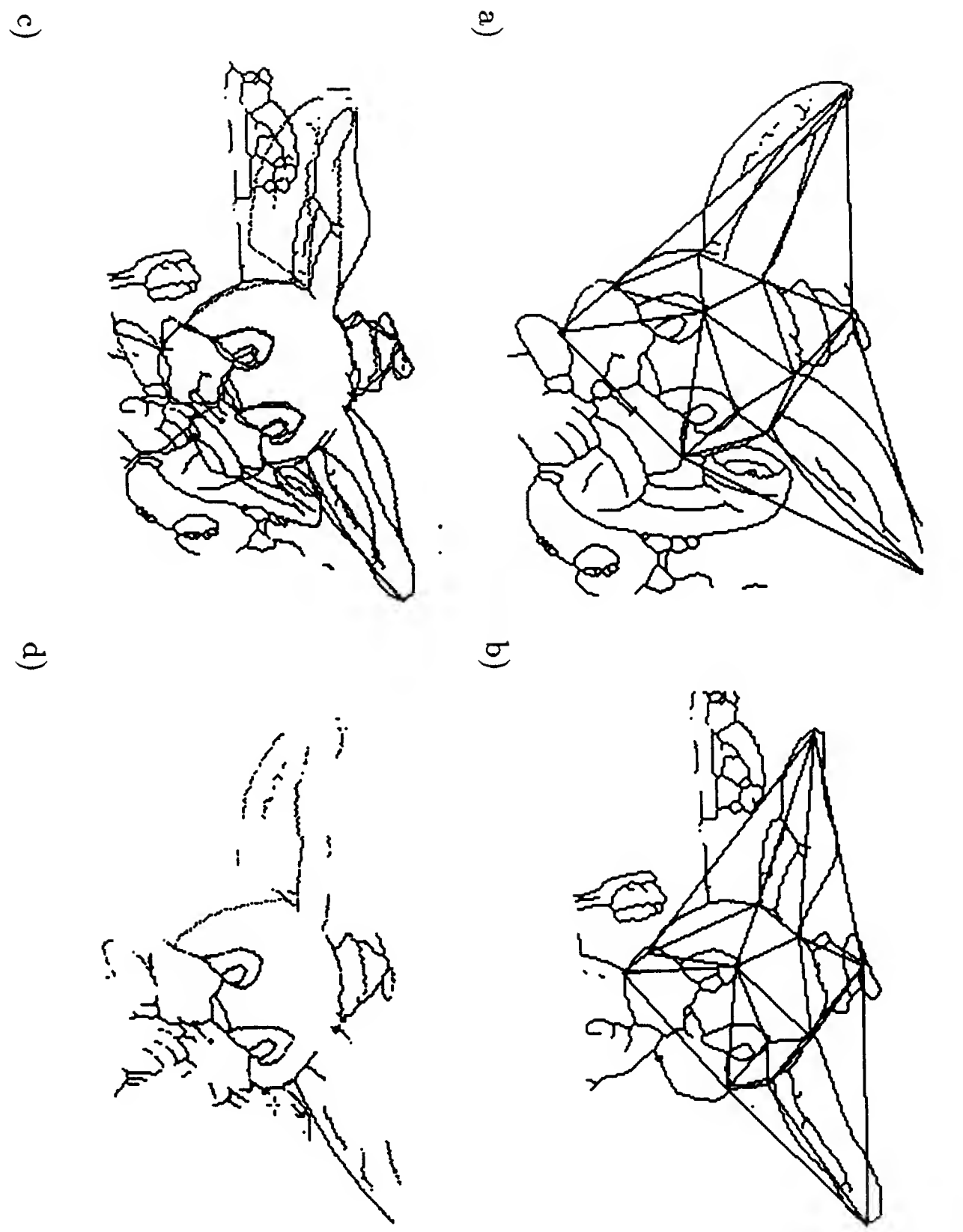

Figure 50. Locally rigid alignment of two images, see the text for an explanation.

7.2 Combining Local Matches

Another means of obtaining a locally rigid approximation to a non-rigid deformation is to combine a set of partial rigid matches together. Thus whereas the triangulation method is "top down", starting with a global rigid approximation and refining it, this method is "bottom up", starting with a set of local rigid approximations and merging them together.

The method starts by computing possible alignments between a model and an image, as described in Chapter 6. The verification stage is modified, however, to allow for partial matches. A partial match occurs if there is good agreement between some part of the model contour and the image, rather than requiring a large percentage of the model contour to be matched. The matched part of the contour must include the feature(s) used to compute the transformation. The rationale for this restriction is that the best part of the match should be near the features that were used to compute the transformation. If there is a good partial match elsewhere on the object, or elsewhere in the image, then it is probably accidental.

A partial match of a model to an image consists of the portions of the model edge contour that are matched, and the corresponding portions of the image edges. Each matched portion of contour is encoded using the locations of its endpoints. The endpoint locations are used to combine partial rigid matches together. An example of a partial match is illustrated in Figure 51. The first part of the figure shows the parts of the image that are matched, with the matching contours shown in bold. The second part of the figure shows the model, with the parts that are matched also displayed in bold. The endpoints of the matched portions of the contour are marked by dots, and the labels A, B, C, D show the correspondence between the image and the model.

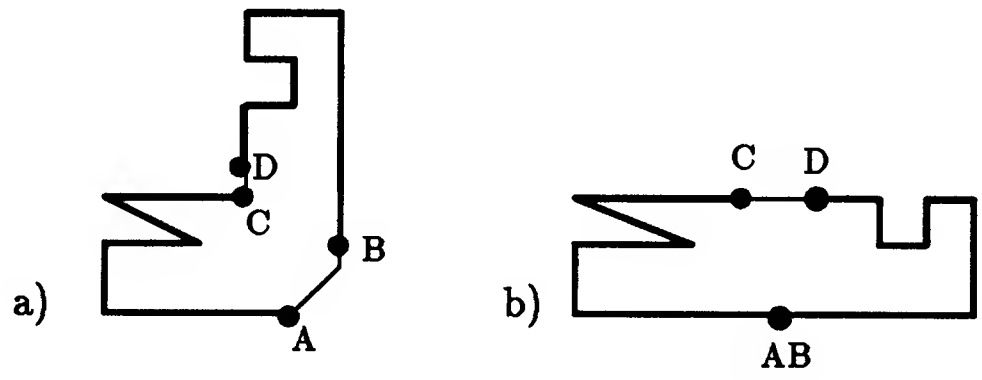

Figure 51. Partial match of a model to an image: a) the partial image match, b) the parts of the model accounted for.

Each partial match defines a correspondence between a portion of a model and an image. If two or more adjacent model parts are also adjacent in the image,

then they form part of a more global, but non-rigid, match. Thus the non-rigid matching algorithm operates by finding sets of partial matches where adjacent model parts are also adjacent in the image. If these partial matches together account for enough of the model, then a non-rigid match is hypothesized.

Adjacency is determined using the endpoints of the matched contour segments. Two partial matches are adjacent if they have one or more segment endpoints that are within some distance ϵ of one another. In the current implementation this distance is a constant 5 pixels. It would probably be better to have some sort of variable threshold.

Adjacent parts of a model are found by entering each partial match (for a given model) into a three-dimensional table indexed by the model coordinates of the matched segments. Any two partial matches that have at least one endpoint within ϵ of one another are defined to be adjacent in the model. Figure 52a illustrates three parts of a model, A, B, and C with two adjacent pairs of parts, (A, B) and (B, C).

Every pair of adjacent model matches is checked for adjacency in the image, using the correspondence between each partial match of the model with the image. In order to be adjacent in the image, at least one of the nearby model endpoints must also be nearby (within ϵ) in the image. In Figure 52b the parts A and B are adjacent in the image, because one endpoint of each part is nearby. The pair (B, C), however, has no nearby endpoints and thus is not adjacent.

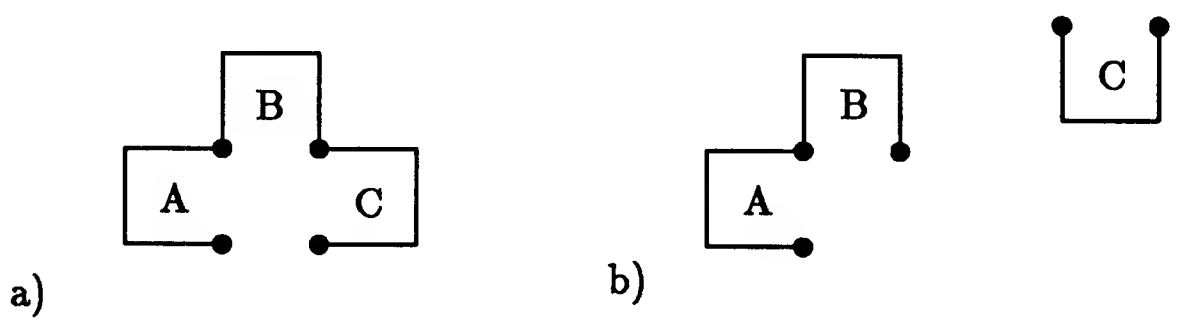

Figure 52. Finding adjacent model and image parts: a) adjacent pairs of model parts, b) only one of which is adjacent in the image.

Once partial matches have been found that are adjacent in both the model and the image, these are combined to determine the total amount of the model accounted for. If it is above a certain percentage, then the match is accepted. Figure 53 shows a match of the laminar object in Figure 46 to a bent instance of the object, with the matched portions of the image contour shown in bold. The match was found using the ORA system, augmented with the partial matching and adjacency computations described here.

Figure 53. Match of a model to a bent instance using the adjacent parts method.

a) b)

Figure 54. The partial matches that comprise the non-rigid match in the previous figure.

The two partial matches that combined to form this match are shown in Figure 54. Each partial match is shown overlaid on the image contours.

The adjacency matching method described in this section first finds all partial matches of a model to an image, and then determines which partial matches are adjacent in both the model and the image. In contrast, recognition systems that use parameterized models generally find a match of one part of a model to an image, and then extend that match by looking for connected matching parts [Brooks81a]. It would be straightforward to similarly make the adjacency matching method use one partial match to limit the search for other partial matches. This, however, introduces a serial dependency of one match on another. On a parallel machine, it is conceivable that all partial matches would be found first, followed by determining which partial matches are adjacent.

7.3 Chapter Summary

The alignment transformation can be used to recognize objects that have been non-rigidly deformed. If an object model specifies a set of rigid subparts, then a separate rigid alignment can be performed for each subpart, subject to the additional constraint that adjacent model parts are also adjacent in the image.

If a model does not specify the rigid subparts of an object, the non-rigid deformation can be recovered at recognition time. Two methods have been presented which approximate a non-rigid deformation by a set of locally rigid alignments. The first method uses triples of image features to tile an image with triangles, and a separate rigid alignment is performed for each triangle. The resulting match is used to find additional corresponding features, and a finer triangulation is computed. The process iterates until either a sufficiently good match is obtained, or the triangles become small.

The second method recovers a non-rigid deformation by combining rigid partial matches. The verification stage of the standard alignment method is modified to allow for partial matches of a model to an image. Partial matches are then combined by checking for adjacent parts of a model that are also adjacent in the image. This method has been implemented and tested on some simple images of non-rigidly deformed objects.

Chapter 8

Human and Machine Recognition

One of the strengths of machine vision research has been the ability to combine independent computational and biological evidence for a given theory [Marr82]. For instance, the development of edge detection operators was based on both computational tractability and physiological plausibility [Marr80]. At higher levels of processing, such as recognition, it has proven more difficult to relate biological systems to computational theories. This chapter considers some psychophysical results that are relevant to the alignment method of recognition. These studies indicate that human recognition is characterized by a degree of tolerance for matching error, and suggest that human recognition may involve separate processes of alignment and comparison.

Unlike model-based recognition systems, human recognition does not seem to simply involve finding a transformation that maps a geometrically accurate model onto an instance in an image. In contrast, human recognition performance is characterized by relatively relaxed matching criteria. For instance, the partial pyramid in Figure 55 cannot actually be a projection of a solid object, even though it appears to be one [Perkins83]. In order to correspond to a real object the three numbered edges must come together at a point, but they do not. Nonetheless, we perceive this as a drawing of a solid object, exhibiting a degree of tolerance in matching.

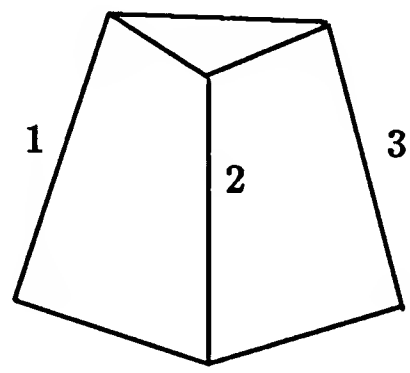

Figure 55. While appearing to be a partial pyramid, this cannot correspond to an actual solid object.

Similarly, in deciding whether cube-like figures are actually possible projections of a cube, people are only about 85% correct [Perkins83]. These errors appear to be due to error tolerance in the recognition process, rather than to limitations

of visual acuity. This is demonstrated by the fact that performance can be greatly improved using special tricks. In order for a figure to be a projection of a cube, the angles about the central vertex must all be at least 90 degrees. Using this rule, people can score nearly 100% correct on the cube classification task [Perkins83]. Thus the human visual system apparently has the ability to extract sufficiently accurate information, however such information is not generally used to judge the correctness of a match.

Having a degree of tolerance in the matching process has two possible benefits in both human and machine recognition. First, it enables a system to handle sensory noise that causes a slight mismatch between a stored model and an instance. Second, it allows a system to use simpler, but somewhat inaccurate methods. For example, the weak perspective imaging model is generally a reasonable approximation to true perspective and thus results in slight errors that would be within tolerance.

While human recognition is more flexible than transforming a geometrically accurate model to match an instance, there is evidence that human recognition involves matching some kind of stored model to an unknown instance. The remainder of this chapter considers some of the psychophysical data that are relevant to the use of an alignment strategy in recognition.

8.1 Alignment in Human Recognition

There is a substantial amount of evidence that people judge the sameness or difference of two objects by first aligning them and then comparing them. These studies either involve mentally rotating one object to match another [Shepard82], or comparing two different images of objects [Rock63]. Some more recent studies show that an alignment strategy is used in recognition tasks as well as in comparison tasks [Jolicoeur85] [Tarr88]. In particular, an alignment strategy is observed in recognition tasks where there are several potentially confusable shapes. An alignment strategy does not appear to be used in tasks where objects can be reliably identified simply based on the presence or absence of a particular feature.

This pattern of results suggests a two-stage matching process, where first viewpoint invariant information is used to hypothesize potential objects, and then alignment is performed when it is necessary to discriminate among more than one possibility. This is similar to the structure proposed by various computational vision researchers (e.g., [Kalvin86]), of an initial viewpoint independent model selection followed by determination of position and orientation.

A number of experiments [Shepard82] have established that in order to

compare a model with an instance, people perform a mental rotation operation. This mental rotation takes time proportional to the degree of orientation difference between the model and the instance. The process apparently involves analog rotation of the object, as evidenced by tasks where people are unexpectedly probed with an instance of the object during the rotation process. If the probe is at the correct intermediate orientation, then recognition of the probe is much more rapid than otherwise.

Another task that exhibits linearly increasing response time with increasing orientation difference is making a judgement about whether or not an instance is a mirror image [Corballis84]. In these experiments, subjects judged whether a two-dimensional character was a mirror image or a normal image. It was found that the identification of a character is relatively independent of orientation, whereas determining whether or not it is a mirror image takes time linear in the difference from an upright orientation. Thus it appears that a character must be rotated to upright before it is possible to determine whether or not it is a mirror image.

At first these results seem to suggest that alignment is not used in recognition of characters, but only in determining mirror reversal after a character has been identified. A follow-up set of experiments shows that there is an orientation effect on letter identification, but it is very rapid, on the order of 15 msec. A set of studies using drawings of natural objects [Jolicoeur85] suggests why the orientation effects are so small in the letter identification task. Recognition times for the drawings were initially linear with the extent of rotation from a canonical position. With practice, however, the effects of orientation become reduced. The practice effects do not, however, transfer to new objects. Thus, it appears that people are learning new models of each object at various orientations. Hence the fairly small rotation effects for letters may be due to the fact that they are highly familiar, and are often seen at other than upright orientations (e.g., books on a shelf, things in a mirror, writing on signs and billboards).

The most complete experiments investigating alignment in human recognition have recently been reported by [Tarr88]. These experiments used novel letter-like characters that contain similar local features. Thus the characters cannot be recognized by simply checking for the presence or absence of a particular feature. Each character has an obvious axis and a clearly marked bottom, as illustrated in Figure 56.

Each character was assigned a name, and was learned in its upright orientation. Subjects were then required to name instances of the characters presented at various orientations. The time to name the character was linearly related

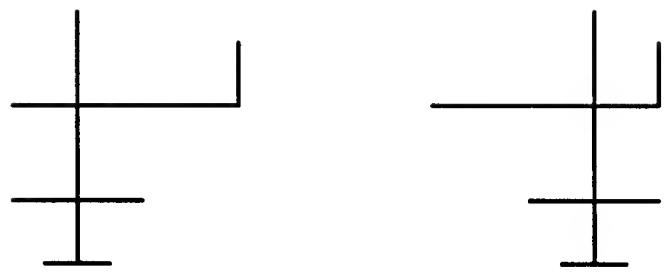

Figure 56. Stimuli used in recognition experiments of [Tarr88].

to the degree of rotation from the upright orientation. With practice recognizing the characters at various orientations, recognition times eventually became nearly the same across all the practiced orientations.

Following practice with the characters at various orientations, novel orientations again produced longer response times. The latencies were linearly related to the difference between the novel orientation and the nearest learned orientation.

These results indicate that for objects that cannot be recognized simply on the basis of presence or absence of a particular feature, people must first align an instance with a stored model. Sufficient experience with a particular orientation of an object causes a model to be formed of the object at that orientation. Once such models have been formed, recognizing an instance of the object at novel orientations requires alignment with the nearest stored model.

Human recognition involves an analog rotation process, and thus can gain a speed advantage by storing multiple models at different orientations. In contrast, the alignment method presented in this thesis is a constant-time operation. Thus the machine alignment method only requires new models for orientations from which different parts of the model are visible. For flat objects, a single model is sufficient.

Overall, the psychophysical data seem to suggest that alignment is an important component of human recognition. While alignment does not always appear to be necessary, it seems necessary whenever the spatial arrangement of the parts of an object is important for recognizing that object. For many objects, the spatial arrangement of parts would seem to be important defining characteristic. Further investigation is required, however, to determine the importance of alignment in recognizing three-dimensional objects.

8.2 3D from 2D Alignment

Human recognition seems to to involve an automatic three-dimensional interpretation of two-dimensional images. Shepard [Shepard82] found that in judging

the size of an object from a line-drawing of its contours, people perform three-dimensional analysis of the shape contours. For instance, of the two blocks shown in Figure 57, people judge the left block to be longer than the right one. However, the two-dimensional length and width of the tops of the two blocks are identical.

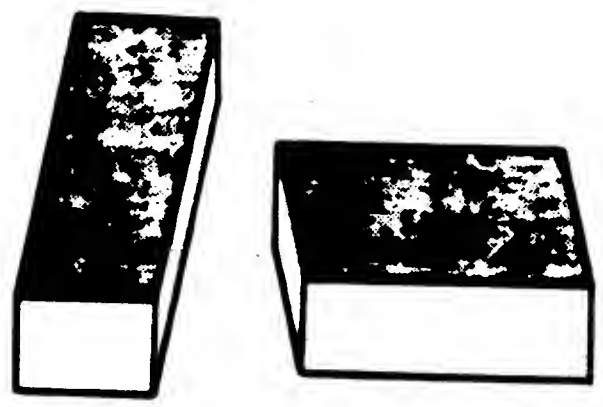

Figure 57. People seem to perform three-dimensional interpretation of the edges in an image.

In fact, these drawings cannot be actual projections of blocks, because the distortion of the different faces is not consistent with a single position and orientation. Having been told this it is still very difficult to believe that the dimensions of the tops are the same. Thus the three-dimensional interpretation appears to happen at a fairly low level.

These results have been interpreted as indicating that three-dimensional shape information is extracted from an image prior to processes such as recognition. Another possibility, however, is that the three-dimensional alignment of certain primitive shapes is used to interpret an image, independent of a particular model. For instance, the primitive shape "right angle" is enough to determine that the right block in Figure 57 is longer than the left one, as illustrated below.

A three-dimensional alignment of a right angle with an image shows how far out of the view plane a surface with a right angle must be rotated in order to result in the image angle. For the block on the left of Figure 57, the front face has right angles, indicating no rotation out of the view plane. Both the top and right faces have non-right angles, however, and have thus been correspondingly rotated out of the image plane. The right face has been rotated extremely, whereas the top one has been rotated relatively little, as illustrated in Figure 58. For the block on the right a similar analysis applies. The front face has right angles, and the top and left faces have both been rotated out of the image plane. However, neither of these faces has been rotated out nearly as much as the right face of the first block.

Figure 58. A surface with a right angles must be rotated out of the plane to yield a given image angle. The block on the left is rotated substantially more than the one on the right.

Thus, performing the three-dimensional alignments to match right angles to the image angles requires the left block to be more lifted out of the plane (more foreshortened) than the right block. Hence, the left block must be longer since the two images are the same size but one object is highly foreshortened. This example illustrates that it is possible to use simple shape primitives to discover three-dimensional alignments without the use of explicit models. Therefore it need not be the case that three-dimensional interpretation happens prior to matching operations.

8.3 Chapter Summary

This chapter has reviewed some of the relevant psychophysical literature on human recognition performance. Unlike machine recognition, human recognition is characterized by a substantial degree of tolerance in matching. This tolerance may be useful for dealing with sensor error, and may also allow relatively simple but somewhat error-prone matching processes to be employed.

Human recognition appears to involve both viewpoint independent matching based on the presence or absence of a few features, and viewpoint dependent matching that recovers the position and orientation of an object. The viewpoint dependent matching process seems to involve an alignment operation, whereby a stored model is transformed such that it matches the input. This transformation process involves analog rotation, similar to that found in mental rotation experiments.

Chapter 9

Summary and Conclusions

This thesis has developed and implemented a method for recognizing solid objects at unknown three-dimensional positions and orientations, from a single two-dimensional view. The objects may be partially occluded, and the image may be highly cluttered.

Most model based recognition systems structure recognition as a search for those transformations that map a large number of model features onto image features. One drawback of this approach is that there can be a substantial chance of falsely finding a match. It is possible to have a number of image features accidentally positioned so that they are consistent with a transformation of a model. The likelihood of this event increases with the simplicity of the features, the amount of clutter in the image, and the difficulty of the recognition task. For example, under projection a given set of model feature points can match many different sets of image feature points.

A second drawback of the traditional approach is the amount of search required to find large sets of features that are consistent with a given transformation. Two techniques are commonly used to find possible transformations. The first method performs a pruned search of the exponential space of possible corresponding model and image features. The second method searches for clusters of similar transformations from a model to an image. In Chapter 3 both of these search techniques were analyzed. The pruned search technique was seen to consider the same transformation a number of times. The generalized Hough transform clustering technique, on the other hand, is prone to finding false clusters unless the clustering table is very large.

Rather than searching for large sets of corresponding features, the recognition method developed in this thesis uses the smallest possible sets of features to hypothesize potential transformations that map a model onto an image. Each transformation is then verified by aligning the model with the image, and comparing the aligned model edge contours with nearby image edge contours. Thus the method uses a different representation to verify transformations than to hypothesize possible transformations.

The central idea underlying the current approach is to separate the matching problem into the two stages of i) finding possible transformations from a model to an image, and ii) checking those transformations. Much of the power of

the approach is derived from the fact that these two operations require relatively different representations of an object. The best representations for computing a transformation are coarse, local and form a relatively sparse description of an object. Features that are coarse and local are more reliably detectable than are fine-scale or global features. Sparse descriptions have fewer features, and thus there are fewer possible matches between a model and an image. The best representations for verification, on the other hand, are more complete but less abstract, such as the edge contours of an object. A complete representation is important for deciding whether a transformation is correct. An abstract representation is not needed because a model has already been transformed into image coordinates.

The secondary ideas underlying the approach are to reduce the amount of search in recognition, and to simplify the problem of representing objects. By identifying the smallest sets of features that are needed to hypothesize a transformation, the method minimizes the number of transformations that must be considered. By representing objects as edge contours and features extracted from edges, the models are very similar to the sensory data. This simplifies the problems of forming models and matching models to images.

A new method of computing a transformation from a solid model to a two-dimensional image was developed in Chapter 4. The method is based on the result that only three corresponding model and image points are needed to define a unique transformation, up to a reflective ambiguity. The computation is simple, only involving solution of two linear systems in two unknowns, and a second order system in two unknowns. Thus the method is fast and relatively robust with respect to noise.

The ORA recognition system, described in Chapter 6, uses this method of computing a transformation to hypothesize possible matches of a model to an image, and then verifies the matches. ORA extracts simple local features from the intensity edges in an image. Each feature defines either three points or a point and an orientation, so only one or two corresponding model and image features are needed to compute a transformation. Thus the system considers at most $O(m^2i^2)$ possible matches for m model features and i image features. Each transformation is verified by aligning the model with the image, and comparing the model edge contours to image edge contours.

The features used for computing transformations are relatively stable over changes in viewpoint, and can be reliably extracted from noisy images. Stability over changes in viewpoint is obtained by segmenting edge contours at zeroes of curvature. These points are preserved under projection, so the resulting segmentation is relatively stable. The features are also local, in order to minimize

sensitivity to partial occlusion.

The verification stage compares the transformed contours of a model against an image. By comparing an entire model with an image, the verification process makes it unlikely that a false match will be accepted. The comparison is hierarchical, first checking local model and image points and orientations, and then checking entire contours. The contour comparison process takes into account both positive and negative evidence of a match. Coterminating model and image edges are strong positive evidence, whereas crossing model and image edges are strong negative evidence.

The basic recognition method has been extended to the case of non-rigid deformations. A non-rigid transformation from a model to an image is approximated by a set of locally rigid transformations. In contrast with existing methods, the locally rigid parts can be recovered at recognition time, rather than requiring them to be specified by the model. Chapter 7 presented two different methods for recovering the rigid subparts of a transformation. The first method is a coarse-to-fine iterative process, and the second method performs a bottom-up combination of partial matches.

The major technical contributions of the recognition method developed in this thesis can be summarized as follows,

- Separating matching into alignment and verification stages allows different information to be used for each stage. Coarse-scale, local, sparse features are the best representation for hypothesizing a transformation. In contrast, for verification the best representation is a more complete but less abstract description, such as the edge contours.
- Rather than searching for large sets of corresponding model and image features, the method uses the smallest possible sets of features to solve for possible transformations, thus minimizing the size of the search space.
- It is shown that three corresponding model and image points determine a unique (up to a reflection) transformation mapping a solid model in three-space onto a two-dimensional image, assuming the weak perspective imaging model.
- A simple, fast, robust method is developed for computing a transformation from three matching model and image points, or from two points and two orientations.
- A shape representation is developed that is relatively insensitive to partial occlusion and stable over different viewpoints. Sensitivity to occlusion is minimized by using local features. Stability over viewpoints is obtained by segmenting edge contours at zeroes of curvature, which are preserved under

projection.

- The verification process compares aligned model edges with image edges, and uses both positive and negative evidence of a match.
- Non-rigid deformations of objects are approximated by a set of local rigid alignments. The rigid parts of an object can either be specified by the object model, or can be recovered dynamically at recognition time.

The separation of recognition into distinct alignment and comparison stages is also supported by psychophysical data, as discussed in Chapter 8. Some recent studies indicate that human recognition involves rotating a stored representation of an object in order to align it with an image [Tarr88] [Jolicoeur88]. One of the strengths of machine vision research has been the ability to combine independent computational and biological evidence for a given theory [Marr82]. At higher levels of processing, such as recognition, this has proven difficult. Thus it is encouraging to find independent support for the importance of alignment in recognition.

Humans appear to employ an alignment strategy in tasks where the spatial configuration of local features is important for recognizing an object, but not when objects can be identified simply based on the presence or absence of a particular feature [Tarr88]. This pattern of results suggests a two-stage matching process, where viewpoint invariant information is used to hypothesize potential objects, and then alignment is performed in cases where more than one object has been hypothesized. This is similar to the structure proposed by various computational vision researchers (e.g., [Kalvin86]), of an initial viewpoint independent model selection followed by determination of position and orientation.

9.1 Future Directions

The recognition method developed in this thesis can be viewed as performing model based segmentation (as can several other methods [Grimson84] [Lamdan87] [Thompson87]), because little interpretation of an image is done before the model matching stage. This type of approach may make the matching part of recognition too difficult, by not first segmenting and classifying parts of an image. That is not to say that the problems in model based vision are not real. Even with better sensory input, it is important to have matching algorithms that are robust, have a low chance of finding a false match, and minimize the amount of search needed to find instances of an object in an image. It is just that the overall importance of these problems in recognition may be exaggerated by the approach.

Unfortunately, attempts at developing methods for segmenting and classifying an image have been relatively unsuccessful in comparison with the model based segmentation approach. One major limitation of these attempts, however, has been the use of one or two images as the starting point for segmentation and classification. Recent suggestions about the use of "active vision" [Aloimonos87] [Bajcsy88] [Ballard88] may lead the way to better input for recognition algorithms, and new visual sensors may provide a computationally tractable means of doing active vision [Mead88].

Recognition methods such as the one developed in this thesis do not require highly accurate grouping mechanisms. Rather, it is only necessary to find parts of an image that are likely to arise from the same object. Groups of features in these regions can then be used to hypothesize potential matches, rather than considering all n-tuples of features in the image.

Appendix A
Computing the Transformation

The lisp code for computing the alignment transformation is included in this appendix. The function `vsub2` is two-dimensional vector subtraction. The functions `pt-x` and `pt-y` select the x and y coordinates of a two-dimensional point. The main function is `3d-alignment-transform`, which takes three two-dimensional image points, `ai`, `bi`, and `ci`, and three three-dimensional model points, `am`, `bm`, and `cm`, and returns a transformation mapping the model to the image.

```
;; Computing the 3d alignment transformation from a point triple
;; returns a list of the model transformation, the image translation
;; b, the scaled rotationmatrix sr, and the scale factor s.
(defun 3d-alignment-transform (ai bi ci am bm cm)
  (destructuring-bind (model-trans bm* cm*)
      ;; transform-model takes three model points and returns the
      ;; transformation that puts am at the origin and bm and cm
      ;; in the z=0 plane.
      (transform-model am bm cm)
    (destructuring-bind (b bi* ci*)
        (translate-image ai bi ci)
      (let* ((l (2d-linear-transform bm* cm* bi* ci*))
             (sr-and-s (3d-rotation-and-scale-from-2d-linear-transform l)))
        (cons model-trans (cons b sr-and-s))))))


(defun translate-image (ai bi ci)
  (let ((trans ai))
    (list trans (vsub2 bi trans) (vsub2 ci trans))))


;; computes a two-dimensional linear transform specified by two
;; corresponding points
(defun 2d-linear-transform (bm cm bi ci)
  (let ((xb (pt-x bm))
        (yb (pt-y bm))
        (xc (pt-x cm))
        (yc (pt-y cm))
        (x1b (pt-x bi))
        (y1b (pt-y bi))
        (x1c (pt-x ci))
```

```
        (y1c (pt-y ci)))
    (let ((det (- (* xc yb) (* xb yc)))
          (l (make-array '(2 2))))
      (if (zerop det)
          (signal 'math:singular-matrix)
          (setf (aref l 0 0) (/ (- (* x1c yb) (* x1b yc)) det))
          (setf (aref l 0 1) (- (/ (- (* x1c xb) (* x1b xc)) det)))
          (setf (aref l 1 0) (/ (- (* y1c yb) (* y1b yc)) det))
          (setf (aref l 1 1) (/ (- (* xc y1b) (* xb y1c)) det))
          l))))


;; computes a three-dimensional scale and rotation from a two-dimensional
;; linear transformation, as described in chapter 4.
(defun 3d-rotation-and-scale-from-2d-linear-transform (l)
  (let ((l11 (aref l 0 0))
        (l12 (aref l 0 1))
        (l21 (aref l 1 0))
        (l22 (aref l 1 1))
        (sr (make-array '(3 3))))
    (let ((w (- (+ (square l12) (square l22))
                (+ (square l11) (square l21))))
      (q (+ (* l11 l12) (* l21 l22))))
  (let* ((c1 (sqrt (* .5 (+ w (sqrt (+ (square w) (* 4 (square q))))))))
         (c2 (if (~zerop c1) (sqrt (abs w)) (/ (- q) c1)))
         (s (sqrt (+ (square l11) (square l21) (square c1)))))
    (setf (aref sr 0 0) l11)
    (setf (aref sr 1 0) l21)
    (setf (aref sr 2 0) c1)
    (setf (aref sr 0 1) l12)
    (setf (aref sr 1 1) l22)
    (setf (aref sr 2 1) c2)
    (setf (aref sr 0 2) (/ (- (* c2 l21) (* c1 l22)) s))
    (setf (aref sr 1 2) (/ (- (* c1 l12) (* c2 l11)) s))
    (setf (aref sr 2 2) (/ (- (* l11 l22) (* l21 l12)) s))
    (list sr s)))))
```

References

1. Aloimonos, Y., Weiss, I. and Bandyopadhyay, A. 1987. Active Vision, *Proc. First International Conference on Computer Vision*, IEEE Computer Society Press, pp. 35-54.

2. Attneave, F. 1954. Some Informational Aspects of Visual Perception, *Psych. Review*, Vol. 61., pp. 183-193.

3. Asada, H. and Brady, M. 1986. The Curvature Primal Sketch, *IEEE Trans. Pat. Anal. and Mach. Intel.*, Vol. 8, No. 1, pp. 2-14.

4. Augusteijn, M.F. and Dyer, C.R. 1986. Recognition and Recovery of the Three-Dimensional Orientation of Planar Point Patterns, *Computer Vision, Graphics and Image Proc.*, Vol. 36, pp. 76-99.

5. Ayache, N. and Faugeras, O.D. 1986. HYPER: A New Approach for the Recognition and Positioning of Two-Dimensional Objects, *IEEE Trans. Pat. Anal. and Mach. Intel.*, Vol. 8, No. 1, pp. 44-54.

6. Bajcsy, R. 1988. Perception with Feedback, *Proc. Image Understanding Workshop*, pp. 279-288, Morgan Kaufmann Publishers, San Mateo, Calif.

7. Ballard, D.H. 1988. Eye Movements and Spatial Cognition, *AAAI Symposium on Phys. and Biol. Approaches to Comput. Vision*, Stanford Univ.

8. Barr, A.H. 1981. Superquadrics and Angle-Preserving Transformations, *IEEE CG and A*, pp. 11-23.

9. Barrow, H.G. and Tenenbaum, J.M. 1976. MSYS: A System for Reasoning About Scenes, SRI AI Center, Tech. Note 121.

10. Basri, R. 1988. The Visual Recognition of Smooth 3D Objects by Alignment, forthcoming Ph.D. thesis, the Weizmann Institute of Science, Israel.

11. Besl, P.J. and Jain, R.C. 1985. Three-Dimensional Object Recognition, *ACM Computing Surveys*, Vol. 17, No. 1 pp. 75-154.

12. Biederman, I. 1985. Human Image Understanding: Recent Research and a Theory, *Comput. Vis., Graphics, and Image Proc.*, Vol. 32, pp. 29-73.

13. Blelloch, G. 1988. Scans as Primitive Parallel Operations. *Proc. Int. Conf. on Parallel Proc.*, 355-362.

14. Blum, H. and Nagel, R.N. 1978. Shape Description Using Weighted Symmetric Axis Features, *Pattern Recognition*, Vol. 10, pp. 167-180.

15. Bolles, R.C. and Cain, R.A. 1982. Recognizing and Locating Partially Visible Objects: The Local Feature Focus Method, *Int. J. Robotics Res.*, Vol. 1, No. 3, pp. 57-82.

16. Bolles, R.C. and Horaud, P. 1986. 3DPO: A Three-Dimensional Part Orientation System, *Int. J. Robotics Res.*, Vol. 5, No. 3, pp. 3-26.

17. Brady, M. and Asada, H. 1984. Smoothed Local Symmetries and Their Implementation, *Int. J. Robotics Res.*, Vol. 3, No. 3, pp. 36-61.

18. Brooks, R.A. 1981. Symbolic Reasoning Around 3-D Models and 2-D Images, *Artificial Intelligence J.* Vol. 17, pp. 285-348.

19. Brooks, R.A. 1981. *Model-Based Computer Vision*, UMI Research Press, Ann Arbor, Mich.

20. Canny, J. 1986. A Computational Approach to Edge Detection, *IEEE Trans. Pat. Anal. and Mach. Intel.*, Vol. 8, No. 6, pp. 34-43.

21. Chin, R.T. and Dyer, C.R. 1986. Model-Based Recognition in Robot Vision, *ACM Computing Surveys*, Vol. 18, No. 1, pp. 67-108.

22. Clemens, D.T. 1986. The Recognition of Two-Dimensional Modeled Objects in Images, Master's Thesis, MIT Dept. of Elect. Eng. and Computer Sci.

23. Corballis M.C. and McLaren, R. 1984. Winding One's Ps and Qs: Mental Rotation and Mirror-Image Discrimination, *J. Exper. Psych.: Human Percep. and Perf.*, Vol. 10, No. 2, pp. 318-327.

24. Curtis, S.R. 1985. Reconstruction of Multidimensional Signals from Zero Crossings, MIT RLE Tech. Report No. 509.

25. Cyganski, D. and Orr, J.A. 1985. Applications of Tensor Theory to Object Recognition and Orientation Determination, *IEEE Trans. on Pat. Anal. and Mach. Intel.*, Vol. PAMI-7, No. 6, pp. 662-673.

26. Davis, L.S. 1979. Shape Matching Using Relaxation Techniques, *IEEE Trans. Pat. Anal. and Mach. Intel.*, Vol. 1, No. 1, pp. 60-72.

27. Duda, R.O. and Hart, P.E. 1973. *Pattern Classification and Scene Analysis*, Wiley, New York.

28. Ettinger, G.J. 1987. Hierarchical Object Recognition Using Libraries of Parameterized Model Sub-Parts, MIT AI Lab Tech. Report No. 963.

29. Feller, W. 1968. *An Introduction to Probability Theory and Its Applications*, Wiley, New York.

30. Fischler, M.A. and Bolles, R.C. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, *Comm. Assoc. Comput. Mach.*, Vol. 24, No. 6., pp. 381-395.

31. Fischler, M.A. and Bolles, R.C. 1986. Perceptual Organization and Curve Partitioning, *IEEE Trans. Pat. Anal. and Mach. Intel.*, Vol. 8, No. 1, pp. 100-104.

32. Efimov, N.V. 1980. *Higher Geometry*, English translation from the Russian, Mir Publishers, Moscow.

33. Fleck, M.M. 1985. Local Rotational Symmetries, MIT Artificial Intell. Lab. Tech. Report, No. 852.

34. Ganapathy, S. 1985. Camera Location Determination Problem, unpublished paper, A.T.&T. Bell Labs.

35. Gigas, Z. and Malik, J. 1988. Computing the Aspect Graph for Line Drawings of Polyhedral Objects, *Proc. IEEE Conf. Robotics and Autom.*, pp. 1560-1566.

36. Goad, C. 1986. Fast 3D Model-Based Vision in *From Pixels to Predicates: Recent Advances in Computational and Robotic Vision*, edited by A.P. Pentland. Ablex, Norwood N.J.

37. Grimson, W.E.L. and Lozano-Pérez, T. 1984. Model-Based Recognition and Localization from Sparse Range or Tactile Data, *Int. J. Robotics Res.*, Vol. 3, No. 3, pp. 3-35.

38. Grimson, W.E.L. and Lozano-Pérez, T. 1987. Localizing Overlapping Parts by Searching the Interpretation Tree, *IEEE Trans. Pat. Anal. and Mach. Intel.*, Vol. 9, No. 4, pp. 469-482.

39. Grimson, W.E.L. 1987. Recognition of Object Families Using Parameterized Models, *Proceedings of the First International Conference on Computer Vision*, IEEE Computer Society Press, pp. 93-100.

40. Grimson, W.E.L. 1988. The Combinatorics of Object Recognition in Cluttered Environments, MIT Artificial Intelligence Laboratory, Memo No. 1019.

41. Herman, M., Kanade, T. and Kurve, S. 1984. Incremental Acquisition of a Three-Dimensional Scene Model from Images, *IEEE Trans. Pat. Anal. and Mach. Intell.*, Vol. 6., No. 3, pp. 331-339.

42. Hillis, W.D. 1986. *The Connection Machine*, MIT Press, Cambridge, Mass.

43. Hoffman, D.D. and Richards, W.A. 1986. Parts of Recognition, in *From Pixels to Predicates: Recent Advances in Computational and Robotic Vision*, edited by A.P. Pentland. Ablex, Norwood N.J.

44. Horn, B.K.P. and Weldon, E.J. 1985. Filtering Closed Curves, *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pp. 478-484.

45. Horn, B.K.P. 1986. *Robot Vision*, MIT Press, Cambridge, Mass.

46. Huttenlocher, D.P. and Ullman, S. 1987. Object Recognition Using Alignment, *Proc. First International Conference on Computer Vision*, IEEE Computer Society Press, pp. 102-111.

47. Ikeuchi, K. 1987. Precompiling a Geometrical Model into an Interpretation Tree for Object Recognition in Bin-Picking Tasks, *Proc. Image Understanding Workshop*, pp. 321-338, Morgan Kaufmann Publishers, San Mateo, Calif.

48. Jolicoeur, P. 1985. The Time to Name Disoriented Natural Objects, *Memory and Cognition*, Vol. 13, No. 4, pp. 289-303.

49. Jacobs, D.W. 1988. The Use of Grouping in Visual Object Recognition, Master's Thesis, MIT Dept. of Elect. Eng. and Computer Sci.

50. Kalvin, A., Schonberg, E., Schwartz, J.T. and Sharir, M. 1986. Two Dimensional Model Based Boundary Matching Using Footprints, *Int. J. Robotics Res.*, Vol. 5, No. 4, pp. 38-55.

51. Kanade, T. and Kender, J.R. 1983. Mapping Image Properties into Shape Constraints: Skewed Symmetry, Affine Transformable Patterns, and the Shape-from-Texture Paradigm, in J. Beck, et. al. (Eds) *Human and Machine Vision*, Academic Press, Orlando, Fla.

52. Koenderink, J.J and VanDoorn, A.J. 1979. Internal Representation of Solid Shape with Respect to Vision, *Biol. Cyber.*, Vol. 32, No. 4, pp. 211-216.

53. Linainmaa, S., Harwood, D. and Davis, L.S. 1985. Pose Determination of a Three-Dimensional Object Using Triangle Pairs, CAR-TR-143, Center for Automation Research, University of Maryland.

54. Lamdan, Y., Schwartz, J.T. and Wolfson, H.J. 1987. On Recognition of 3-D Objects from 2-D Images, New York University, Courant Institute Robotics Report, No. 122.

55. Little, J.J., Blelloch, G. and Cass, T. 1987. Parallel Algorithms for Vision on the Connection Machine, *Proceedings of the First International Conference on Computer Vision*, IEEE Computer Society Press, pp. 587-591.

56. Lowe, D.G. 1985. *Perceptual Organization and Visual Recognition*, Kluwer Academic Publishers, Hingham, Mass.

57. Lowe, D.G. 1987. Three-Dimensional Object Recognition from Single Two-Dimensional Images, *Artificial Intelligence J*, Vol. 31, pp. 355-395.

58. Mahoney, J.V. 1987. Image Chunking: Defining Spatial Building Blocks for Scene Analysis, MIT Artificial Intelligence Lab., Tech. Report 980.

59. Marr, D. and Nishihara, H.K. 1978. Representation and Recognition of the Spatial Organization of Three-Dimensional Shapes, *Proc. Royal Society Lon.*, B200, pp. 269-294.

60. Marr, D. and Hildreth, E. 1980. Theory of Edge Detection, *Proc. Royal Society Lon.*, B207, pp. 187-217.

61. Marr, D. 1982. *Vision* Freeman, San Francisco.

62. Mead, C.A. 1988. Analog VLSI Models of Neural Systems *AAAI Symposium on Phys. and Biol. Approaches to Comput. Vision*, Stanford Univ.

63. Mokhtarian, F. and Mackworth, A. 1986. Scale-Based Description and Recognition of Planar Curves and Two-Dimensional Shapes, *IEEE Trans. Pat. Anal. and Mach. Intel.*, Vol. 8, No. 1.

64. Nevatia, R. and Binford, T.O. 1977. Description and Recognition of Curved Objects, *Artificial Intell. J.*, Vol. 8., pp. 77-98.

65. Pentland, A. 1985. Perceptual Organization and the Representation of Natural Form, SRI International Tech. Note No. 357.

66. Perkins, W.A. 1977. A Model-Based Vision System for Scenes Containing Multiple Parts, *Proc. 5th Int. Conf. on Artif. Intell.*, pp. 678-684.

67. Perkins, D.N. 1983. Why the Human Perceiver Is a Bad Machine, in *Human and Machine Vision* edited by J. Beck, B. Hope and A. Rosenfeld, pp. 341-364. Academic Press, Orlando, Fla.

68. Poggio, T. 1988. The MIT Vision Machine *Proc. of the Image Understanding Workshop*, pp. 177-198, Morgan Kaufmann Publishers, San Mateo, Calif.

69. Preparata, F.P. and Shamos, M.I. 1985. *Computational Geometry An Introduction.* Springer-Verlag, New York.

70. Roberts, L.G. 1965. Machine Perception of Three-Dimensional Solids, in Tippett et. al. (Eds) *Optical and Electro-Optical Information Processing*, MIT Press, Cambridge, Mass.

71. Rock, I. and Leaman, R. 1963. An Experimental Analysis of Visual Symmetry, *Acta Psychologica*, Vol. 23., pp. 171-183.

72. Rosenfeld, A., Hummel, R. and Zucker, S. 1976. Scene Labeling by Relaxation Operations, *IEEE Trans. Sys., Man, Cyber.*, Vol. 7, pp. 420-433.

73. Sadaji, F.A. 1980. Three-Dimensional Moment Invariants, *IEEE Trans. Pat. Anal. Mach. Intel.*, Vol. 2, pp. 127-136.

74. Saund, E. 1988. Symbolic Construction of a 2D Scalespace Image, unpublished paper, MIT AI Laboratory.

75. Shepard, R.N. and Cooper, L.A. 1982. *Mental Images and Their Transformations.* MIT Press, Cambridge, Mass.

76. Silberberg, T.M., Harwood, D.A. and Davis, L.S. 1986. Object Recognition Using Oriented Model Points, *Computer Vision, Graphics, and Image Proc.*, Vol. 35, pp. 47-71.

77. Schwartz, J.T. and Sharir, M. 1987. Identification of Partially Obscured Objects in Two and Three Dimensions by Matching of Noisy Characteristic Curves, *Int. J. Robotics Res.*, Vol. 6, No. 2, pp. 29-44.

78. Tarr, M.J. and Pinker, S. 1988. Mental Rotation and Orientation-Dependence in Shape Recognition, unpublished paper, MIT Dept. of Brain and Cognitive Science.

79. Teague, M.R. 1980. Image Analysis via the General Theory of Moments, *J. Opt. Soc. Amer.*, Vol. 70, pp. 920-930.

80. Thompson, D.W. and Mundy, J.L. 1987. Three-Dimensional Model Matching from an Unconstrained Viewpoint, *Proceedings of the International*

Conference on Robotics and Automation, IEEE Computer Society Press, pp. 208-220.

81. Turney, J.L. Mudge, T.N. and Voltz, R.A. 1985. Recognizing Partially Occluded Parts, *IEEE Trans. Pat. Anal. and Mach. Intel.*, Vol. 7, No. 4, pp. 410-421.

82. Ullman, S. 1987. An Approach to Object Recognition: Aligning Pictorial Descriptions, MIT Artificial Intelligence Lab., Memo No. 931.

83. VanHove, P. 1987. Model-Based Silhouette Recognition, *Proceedings of the IEEE Computer Society Workshop on Computer Vision.*

84. Watt, R.J. and Morgan, M.J. 1985. A Theory of the Primitive Spatial Code in Human Vision, *Vision Res.*, Vol. 25, No. 11, pp. 1661-1674.

85. Witkin, A.P. and Tenenbaum, J.M. 1983. On the Role of Structure in Vision, in *Human and Machine Vision* edited by J. Beck, B. Hope and A. Rosenfeld, pp. 481-544. Academic Press, Orlando, Fla.

86. Witkin, A.P. 1985. Scale-Space Filtering, in *From Pixels to Predicates: Recent Advances in Computational and Robotic Vision*, edited by A.P. Pentland, pp. 5-19. Ablex, Norwood, N.J.

87. Yuille, A.L. and Poggio, T. 1986. Scaling Theorems for Zero Crossings, *IEEE Trans. Pat. Anal. and Mach. Intell.*, Vol. 8, No. 1, pp. 15-25.

88. Yale, P.B. 1968. *Geometry and Symmetry*, Holden-Day Publishers, San Francisco.